Predicting Student Performance in an Intelligent Tutoring System

A thesis submitted for the degree of
Doctor of Natural Science (Dr. rer. nat.)

by

Nguyen Thai-Nghe

Department of Computer Science
Information Systems and Machine Learning Lab (ISMLL)
University of Hildesheim, Germany

Winter 2011

To my parents, my wife, and my daughter

Acknowledgements

First of all, I would like to express the most profound gratitude to my supervisors Prof. Dr. Dr. Lars Schmidt-Thieme and Dr. Tomáš Horváth, who have given me invaluable advices and feedbacks. I am very happy to work with them and I have learned many knowledge that are very essential for my research.

I would like to thank Prof. Dr. Neil T. Heffernan who is the second referee of this thesis.

I would also thank to my good friends - members from the ISMLL group. We have a good time period working together, not only in research but also in daily-life activities. I especially thank Lucas Drumond for his fruitful comments.

I would like to thank the Cantho University - Vietnam, especially the TRIG project for supporting me a scholarship to study in Germany. Many thanks also go to the ISMLL for supporting me a fellowship as well as funding for all my trips to many conferences. Moreover, I would like to thank the DAAD (Deutscher Akademischer Austausch Dienst) for supporting me one-year fellowship.

Finally, I would like to express my deep gratitude to my grandparents and my parents who have grown up and educated me. I also would like to thank my brothers and my sisters who have supported me both materials and spirits. Last but not least I would like to thank my wife and my daughter who give me love, strength, and encouragement.

Abstract

Predicting student performance (PSP) is an important task in Student Modeling where we would like to know whether the students solve the given problems (tasks) correctly, so that we can understand how the students learn, provide them early feedbacks, and help them getting better in studying.

This thesis introduces several approaches, which mainly base on state-of-the-art techniques in Recommender Systems (RS), for student modeling, especially for PSP.

First, we formulate the PSP problem and show how to map this problem to rating prediction task in RS and to forecasting problem.

Second, we propose using latent factor models, e.g., matrix factorization, for student modeling. These models could implicitly take into account the student and task latent factors (e.g., slip and guess) as well as student effect/bias and task effect/bias. Moreover, there is a fact that "similar students" may have "similar performances", we suggest using k-nearest-neighbors collaborative filtering to take into account the correlations between the students and the tasks.

Third, in student's problem solving, each student performs several tasks, and each task requires one or many skills, while the students are also required to master the skills that they have learned. We propose to exploit such multiple relationships by using multi-relational matrix factorization approach.

Fourth, as the student performance (student knowledge) cumulates and improves over time, a "trend line" could be observed in his/her performance. Similar to time series, for solving this problem, forecasting techniques would be reasonable choices. Furthermore, it is well-know that student (human) knowledge is diverse, thus, thought and performance of one student may differ from another one. To cope with these aspects, we propose personalized forecasting methods which use the past performances of individual student to forecast his/her own future performance.

Fifth, since student knowledge changes over time, temporal/sequential information would be an important factor in PSP. We propose tensor factorization methods to model both the student/task latent factors and the sequential/temporal effects.

Sixth, we open an issue for recommendation in e-learning, that is, recommending the tasks to the students. This approach can tackle existing issues

in the literature since we can recommend the tasks to the students using their "performance" instead of their "preference". Based on student performance, we can recommend suitable tasks to the students by filtering out the tasks that are too easy or too hard, or both, depending on the system goal. Furthermore, we propose using context-aware factorization approach to utilize multiple interactions between the students and the tasks.

Seventh, we discover a characteristic in student performance data, namely class imbalance problem, i.e., the number of correct solutions are higher than the number of incorrect solutions, which may hinder classifiers' performance. To tackle this problem, we introduce several methods as well as introducing a new evaluation measure for learning from imbalanced data.

Finally, we validate the proposed methods by many experiments. We compare them with other state-of-the-art methods and empirically show that, in most of the cases, the proposed methods can improve the prediction results. We therefore conclude that our approaches would be reasonable choices for student modeling, especially for predicting student performance.

Last but not least, we raise some open issues for the future research in this area.

Contents

1 Introduction **1**
- 1.1 Introduction . . . 1
  - 1.1.1 Intelligent Tutoring Systems (ITSs) . . . 1
  - 1.1.2 Recommender Systems (RSs) . . . 4
  - 1.1.3 Predicting Student Performance in an ITS . . . 5
- 1.2 Contribution . . . 6
- 1.3 Published Work . . . 9
- 1.4 Chapter Overview . . . 10

2 Problem Definition **12**
- 2.1 Problem Formulation . . . 12
- 2.2 Examples of Predicting Student Performance . . . 15
  - 2.2.1 Predicting Student Performance in an ITS . . . 16
  - 2.2.2 Predicting Student Performance in an Academic System . . . 17

3 Data sets and Pre-Processing **18**
- 3.1 Data Sets . . . 18
  - 3.1.1 Cognitive Tutor Data (KDD Cup 2010 Data) . . . 19
  - 3.1.2 ASSISTments Platform Data . . . 20
- 3.2 Data Mapping for Collaborative Filtering . . . 21
- 3.3 Data Mapping for Regression Problem . . . 24
- 3.4 Data Mapping for Classification Problem . . . 24
- 3.5 Data Encoding and Dealing with Multiple Skills . . . 25
- 3.6 Data Splitting Protocol . . . 26

4 State-of-the-art Methods and Related Work **27**
- 4.1 Classification and Regression Methods . . . 27
- 4.2 Bayesian Knowledge Tracing (BKT) . . . 30
- 4.3 Performance Factors Analysis (PFA) . . . 35
- 4.4 Selection of the State-of-the-art Methods . . . 36
- 4.5 Factorization Techniques . . . 38

5 Student Modeling with Latent Factor Models **39**
5.1 Introduction . . . 39
5.2 Modeling Student/Task Latent Factors . . . 41
5.2.1 Training Phase . . . 42
5.2.2 Prediction Phase . . . 44
5.2.3 An Example . . . 44
5.3 Modeling Student/Task Effects (Biases) . . . 45
5.4 Modeling Student/Task Correlations . . . 47
5.4.1 User-based Collaborative Filtering (User-kNN) . . . 48
5.4.2 Item-based Collaborative Filtering (Item-kNN) . . . 48
5.5 Experiments . . . 49
5.5.1 Software implementations . . . 49
5.5.2 Experimental Setting . . . 50
5.5.3 Experimental Results . . . 52
5.6 Discussion . . . 56

6 Exploiting Multi-Relational Aspects in Student Modeling **57**
6.1 Problem Reformulation for Multi-relational Data . . . 58
6.2 Student Modeling with Multi-Relational Matrix Factorization (MRMF) 60
6.3 Student Modeling with Weighted Multi-Relational Matrix Factorization (WMRMF) . . . 61
6.4 Experiments . . . 63
6.4.1 Entity Relationship Diagram Revisions . . . 63
6.4.2 Experimental Setting . . . 63
6.4.3 Experimental Results . . . 64
6.5 Conclusion . . . 65

7 Personalized Forecasting Student Performance **67**
7.1 Classical Forecasting Techniques . . . 68
7.2 Personalized Forecasting Techniques . . . 70
7.2.1 Strategies for Using Historical Data . . . 70
7.2.2 Non-Biased Personalized Forecasting . . . 72
7.2.3 Personalized Forecasting with "Student-Task-Biases" . . . 74
7.2.4 Personalized Forecasting with "Discounted-Mean" . . . 76
7.3 Bootstrapping and k-Step-Ahead Forecasting . . . 77
7.4 Experiments . . . 77
7.4.1 Experimental Setting . . . 78
7.4.2 Experimental Results . . . 79
7.5 Conclusions . . . 81

8 Temporal Effects in Student Modeling **83**
8.1 Problem Formulation Extension 84
8.2 Canonical Tensor Factorization Models 85
8.3 Tensor Factorization - Averaging Approach (TFA) 86
8.4 Tensor Factorization - Weighting Approach (TFW) 87
8.5 Tensor Factorization - Moving Average Forecasting (TFMAF) 87
8.6 Tensor Factorization Forecasting (TFF) 88
8.7 Experiments 90
8.7.1 Experimental Setting 90
8.7.2 Experimental Results 91
8.8 Conclusion 93

9 Context-Aware Factorization Models for Student's Task Recommendation **94**
9.1 Introduction 94
9.2 Related Work 96
9.3 Problem Reformulation 96
9.4 None-Context Approach (Baseline) 97
9.5 Context-Aware Factorization Models 97
9.5.1 Pre-Filtering 98
9.5.2 Contextual Modeling 98
9.6 Experiments 98
9.6.1 Data Sets 98
9.6.2 Experimental Setting 99
9.6.3 Experimental Results 99
9.7 Discussion 100

10 Learning from Imbalanced Data **101**
10.1 Introduction 102
10.2 Problem Formulation 103
10.3 State-of-the-art Methods and Related Work 103
10.3.1 Data-Level Approach 104
10.3.2 Classifier-Level Approach 106
10.3.3 Combination Approach 107
10.3.4 Evaluation Measures 109
10.4 Compound Cost-Sensitive Learning Methods 112
10.4.1 Combining Sampling Techniques with CSL (S-CSL) 112
10.4.2 Optimized Cost Ratio for CSL (O-CSL) 115
10.5 Thresholding on Bayesian Posterior Probabilities 117
10.5.1 Learning in Bayesian Networks 118
10.5.2 Thresholding on Bayesian Posterior Probabilities 119
10.5.3 Thresholding on Sampling Data 121
10.6 B42 - A New Evaluation Measure 124

10.7 Experiments 125
10.7.1 Data Sets 126
10.7.2 Experimental Setting 128
10.7.3 Experimental Results 129
10.8 Conclusion 146

11 Conclusion **147**
11.1 Summary 147
11.2 Discussion 151
11.3 Future Direction 154

Index **162**

References **179**

List of Figures

1.1 Constitution of an ITS (picture source: Woolf (2008)) 2
1.2 Main components of an Intelligent Tutoring System 3
1.3 "Similarities" between E-learning systems and Recommender Systems . 4

2.1 Mapping predicting student performance as rating prediction in Recommender Systems (p: performance; r: rating) 14
2.2 Mapping predicting student performance as forecasting problem 15
2.3 An example of predicting student performance in an ITS (picture is adapted from https://pslcdatashop.web.cmu.edu/KDDCup). 16
2.4 An example of predicting student performance in an Academic System . 17

3.1 A snapshot of the KDD Cup 2010 data sets 20
3.2 A snapshot of the ASSISTments data set 21
3.3 Data splitting protocol (source: KDD Cup 2010) 26

4.1 Bayesian Knowledge Tracing as an Hidden Markov Model 30
4.2 Bayesian Knowledge Tracing and Prior Per Student (picture source: Pardos & Heffernan (2010)) 34

5.1 An example of factorizing on students and tasks 45
5.2 RMSE: Latent factor models vs. others (solving-step is used as task) . . 54
5.3 RMSE: Latent factor models vs. others (skill is used as task) 54

6.1 ERDs present using a single relation for matrix factorization 58
6.2 Entity relationship diagram includes useful information for PSP 59
6.3 Examples of matrix representations (p is a performance score, e.g. $p \in [0..1]$) 59
6.4 Entity relationship diagrams are used for experiments 63
6.5 Using Student-Performs-Task as the main relation: RMSE of (W)MRMF vs. the other models 64
6.6 Using Student-Applies-Skill as the main relation: RMSE of (W)MRMF vs. the state-of-the-art BKT models 65

7.1 An illustration of forecasting at time t 69
7.2 An example of using personalization in PSP (*global model*: Using information (performances) of all other students to predict / forecast Tom's performance; and *individual model*: Using Tom's performance to predict / forecast himself) 71
7.3 An illustration of personalized forecasting which uses all historical data controlled by the history length L, or using all historical data in the same unit and section. This illustration describes for one student. 71
7.4 RMSE: *histLength* vs. *secLength* 79
7.5 Sequential effect on the student performance: $y - axis$ is the average of correct performance and $x - axis$ is the sequence of problems (ID) aggregated from the steps. Typical results of Unit 1 and Section 1 . . . 80
7.6 RMSE: Personalized Forecasting vs. other methods 81
7.7 RMSE: Personalized Forecasting vs. Bayesian Knowledge Tracing 81

8.1 CANDECOM-PARAFAC method: A tensor is decomposed into three low-rank matrices. 86
8.2 Temporal effect on student performance 91
8.3 RMSE: Modeling temporal effect using tensor factorization 92

9.1 An illustration of None-Context vs. Context-aware approach 97

10.1 Bayesian Network types 118
10.2 Symmetric and Asymmetric Beta Distributions 125
10.3 Distribution of columns and %minority examples on Netflix data set . . 128
10.4 Cost ratio and total cost relationship for typical results 130
10.5 Average of TPR and GMean: NB (leftmost), . . . , EnsBNOpt (rightmost) 134
10.6 Cost weight distribution of the AUC (on nf-05p data set), B42, and H . 143
10.7 Typical results of the AUC, the True positive rate, and the B42 145

List of Tables

3.1 Information of Original Data Sets 19
3.2 Data sets' attributes in our notations 22
3.3 Mapping student performance data to User-Item-Rating 23
3.4 Class imbalance information in student performance data 25

4.1 The CPD of learn rate: $P(L_t|L_{t-1})$ 31
4.2 The CPD of $P(X_t|L_t)$ 31

5.1 RMSE of using different sets of attributes as items (tasks) 53
5.2 RMSE: Collaborative Filtering Approaches vs. Bayesian Knowledge Tracing 55
5.3 Hyper parameters and running-time approximation (minutes) 55

6.1 Entity information from three data sets 64
6.2 Relation information from three data sets 64
6.3 Hyper parameters are used for experiments. 66

7.1 Comparison of two weighting approaches 77
7.2 RMSE: Personalized Forecasting (N-PSEF, B-PSEF) vs. Classical (non-personalized) Forecasting (SEF, DEF) 80

8.1 RMSE: Bayesian Knowledge Tracing vs. Tensor Factorization Models . 92
8.2 Hyper parameters of the tensor factorization methods 93

9.1 MAE: None-Context vs. Context-Aware Methods 99
9.2 RMSE: None-Context vs. Context-Aware Methods 100

10.1 Confusion matrix 106
10.2 Cost matrix 106
10.3 Simplified cost matrix 110
10.4 Data sets 127
10.5 S-CSL: Average costs on other data sets 131
10.6 S-CSL: Average costs on Educational Data 132

10.7 O-CSL: Average costs on educational data 132
10.8 O-CSL: GMean on educational data 133
10.9 O-CSL: TPR on educational data 133
10.10O-CSL: Results of GMean on other data sets 133
10.11 Details of F1-Measure and Average of other Metrics 136
10.12 Thresholding on Bayesian Posterior Probabilities - Educational data . . 137
10.13 Detailed results of F1-Measure and average of other Metrics 139
10.14 Average results: Thresholding on Sampling Data 140
10.15 Thresholding on sampling data (using SMOTE) 141
10.16 Thresholding on Sampling Data (using TLINK) 142
10.17 Results of ℓ_2-SVM (base) and ℓ_2-LR on B42, AUC, and H 144
10.18 Win/tie/lose results aggregated from Table 10.17 to 3 groups; ℓ_2-SVM (base) vs. ℓ_2-LR 145
10.19 The B42 disagrees with the AUC 17 times out of 36 data sets 145
10.20 The B42 disagrees with the H 8 times out of 36 data sets 145

11.1 Overall comparison: Using solving-step as an item (the methods are ranked by using RMSE) 149
11.2 Overall comparison: Using skill (KC) as an item (the methods are ranked by using RMSE) 151
11.3 RMSE: Overall comparison on validation sets, using solving-step as an item 153
11.4 RMSE: Overall comparison on validation sets, using skill as an item . . 153
11.5 RMSE on the leaderboard - KDD Challenge 2010 - Best student teams . 154

Notation

This thesis will adhere to the following notational conventions:

S	a set of students (users)
s	student s
I	a set of tasks (items)
i	task i
P	a set of performance scores (ratings)
p or p_{si}	an actual performance of student s on task i
$\hat{p}$ or $\hat{p}_{si}$	a predicted performance of student s on task i
$\mathbf{X}$ or $\mathbf{W}$	a matrix, denoted by a capital bold letter
$\mathbf{X}^T$ or $\mathbf{W}^T$	a transposed matrix
w_k	a vector (one index)
w_k^T	a transposed vector
w_{uk}	an element of a vector (two indices)
$\mathcal{Z}$	a tensor, denoted by a mathcal letter
$\circ$	an outer product

We interchangeably call student, task, and performance as user, item, and rating, respectively.

Abbreviation

BKT	Bayesian Knowledge Tracing
BKT-BF	Bayesian Knowledge Tracing - Brute Force
BKT-EM	Bayesian Knowledge Tracing - Expectation Maximization
BMF	Biased Matrix Factorization
B-PDEF	Biased Personalized Double Exponential smoothing Forecaster
B-PDMF	Biased Personalized Discounted-Mean Forecaster
B-PSEF	Biased Personalized Single Exponential smoothing Forecaster
CF	Collaborative Filtering
CFA	Correct First Attempt
CFAR	Correct First Attempt Rates
CSL	Cost-Sensitive Learning
DEF	Double Exponential smoothing Forecaster
ITS	Intelligent Tutoring Systems
kNN	k-Nearest Neighbors
MAE	Mean Absolute Error
MF	Matrix Factorization
MRMF	Multi-Relational Matrix Factorization
N-PSEF	Non-biased Personalized Single Exponential smoothing Forecaster
N-PDEF	Non-biased Personalized Double Exponential smoothing Forecaster
O-CSL	Optimized cost ratio for Cost-Sensitive Learning
PSP	Predicting Student Performance
RMSE	Root Mean Squared Error
RS	Recommender Systems
ROS	Random Over-Sampling
RUS	Random Under-Sampling
S-CSL	combining Sampling with Cost-Sensitive Learning
SEF	Single Exponential smoothing Forecaster
SGD	Stochastic Gradient Descent
SMOTE	Synthetic Minority Oversampling Technique
TFA	Tensor Factorization - Averaging approach
TFF	Tensor Factorization Forecasting
TFMAF	Tensor Factorization Moving Average Forecasting
TFW	Tensor Factorization - Weighting approach
TLINK	Tomek's Link (an approach for cleaning data)
WMRMF	Weighted Multi-Relational Matrix Factorization

Chapter 1

Introduction

Contents

1.1 Introduction 1
1.1.1 Intelligent Tutoring Systems (ITSs) 1
1.1.2 Recommender Systems (RSs) 4
1.1.3 Predicting Student Performance in an ITS 5
1.2 Contribution 6
1.3 Published Work 9
1.4 Chapter Overview 10

1.1 Introduction

Computer systems have been used for educational purposes since the early 1960s (Corbett *et al.*, 1997). Intelligent Tutoring Systems (ITSs) are such systems, which can effectively help both teachers and students getting better in teaching and learning. Since this thesis is partly involved in the ITSs, we briefly introduce them in the following section.

1.1.1 Intelligent Tutoring Systems (ITSs)

Intelligent tutoring systems have been shown to be highly effective in helping the students learning better. For example, Shute *et al.* (1989) claimed that the students using an ITS for economics could perform equally well as the students taking a traditional economics course, but required half as much time covering the materials (Beck *et al.*, 1996).

Indeed, the ITSs are formed by three fields: Computer Science, Psychology, and Education, as illustrated in Figure 1.1, in which, (i) Artificial Intelligence addresses how to reason about intelligence and thus learning, (ii) Psychology (Cognitive Science)

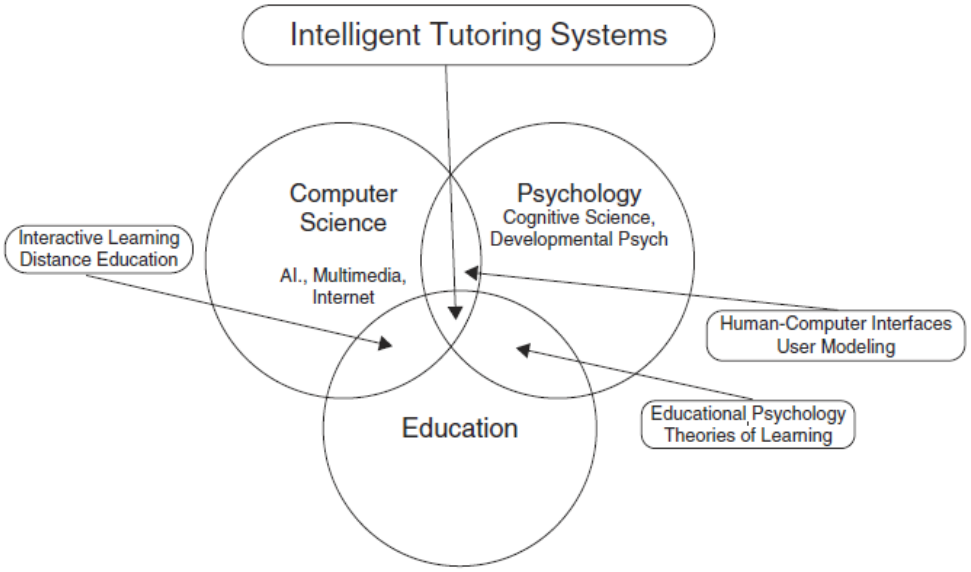


Figure 1.1: Constitution of an ITS (picture source: Woolf (2008))

addresses how people think and learn, and (iii) Education focuses on how to best support teaching/learning (Woolf, 2008).

Although different ITSs may have different structures, e.g., in (Corbett *et al.*, 1997; Freedman *et al.*, 2000; Massey *et al.*, 1988; Woolf, 2008), the basic structure of an ITS has four components (modules/models): Student Model, Tutoring/Instructor Model, Domain Model, and User Interface, as presented in Figure 1.2.

In this thesis we focus on using Machine Learning, which is a branch of Artificial Intelligence (Duda *et al.*, 2000), for modeling a part of the *Student Model*, namely predicting student performance (PSP). However, for later referencing, we briefly introduce all of four components of an ITS, as presented in Figure 1.2 (see Woolf (2008) for details).

1. **Student Model**: The student model tracks information of each individual student (e.g., tracking possible misconceptions, time spent on problems, hints requested, correct answers, and preferred learning style, etc). This model tracks how well a student is performing on the task (material) being taught and represents students' mastery of the domain (e.g. typical student skills) (Beck *et al.*, 1996; Woolf, 2008).

2. **Domain Model** (or Domain Knowledge): This model represents the expert knowledge, or how experts perform in the domain. It contains the information that the tutor will use to teach the students. For example, it includes definitions, processes, or skills needed to multiply numbers, generate algebra equations, etc (Woolf, 2008).

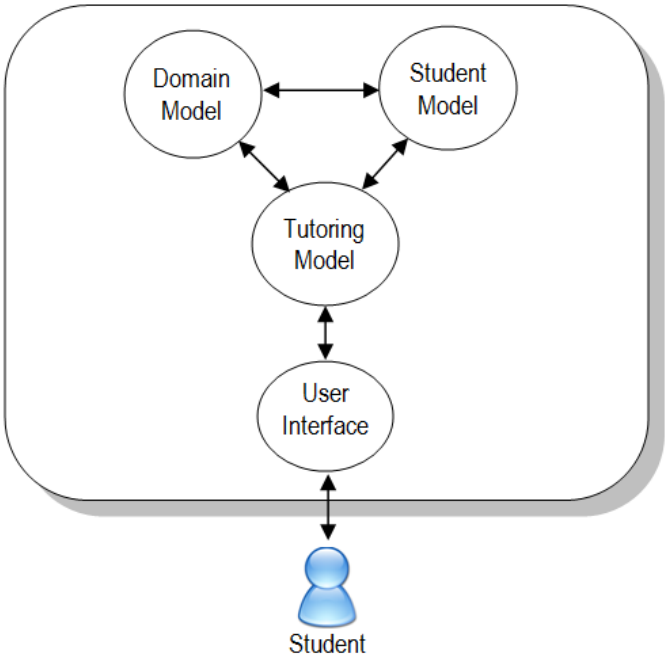


Figure 1.2: Main components of an Intelligent Tutoring System

3. **Tutoring/Instructor Model** (or Pedagogical Module): This module represents teaching processes/strategies (examples and analogies). For example, information about when to review, when to present a new topic, and which topic to present is controlled by this pedagogical module. The student model is used as input to this module, so the pedagogical decisions reflect different needs of each student (Beck *et al.*, 1996).

4. **User Interface** (or Communication Module/Learning Environment): This module presents the methods for interacting between the students and the systems, e.g. via graphical interfaces, animated agents, dialogs, screen layouts, etc. An important problem in this module is how the tasks (materials/learning objects) should be presented to the students in the most effective way.

Indeed, Student Modeling in the ITSs has been taken into account over the years. One of the student modeling tasks is to trace the student's knowledge, and thus student's performance (student performance). According to Corbett & Anderson (1995), the student's knowledge state is not fixed, but is assumed to be increasing. Thus, the authors have incorporated a Bayesian statistical model into the tutors for tracing the student knowledge, and eventually predicting student performance. This model was called Knowledge Tracing which now becomes the state-of-the-art method in student modeling and, as we will show in Chapter 4, the number of papers which have been

proposed to improve it increase over years.

In this work, we will bring a new approach for student modeling, which is based on Recommender Systems, as briefly introduced in the following.

1.1.2 Recommender Systems (RSs)

Recommender systems are widely used in many areas, especially in e-commerce (Jannach *et al.*, 2011; Ricci *et al.*, 2011; Schafer *et al.*, 1999). They belong to information filtering systems which aim at making vast catalogs of items (e.g. movies, television programs, musics, web pages etc.) consumable by learning user preferences and to apply them to the items formerly unknown to the user. Thus they can learn which items have a high likelihood of being interesting to the target user. The most famous recommender system and indeed one of the first commercial recommender system is the Amazon's "Customers Who Bought This Item Also Bought" (Linden *et al.*, 2003).

The two most common tasks in recommender systems are Top-N *item recommendation* where the recommender suggests a ranked list of (at most) N items to a user and *rating prediction* where the aim is to predict the preference score (rating) for a given user-item combination. For item recommendation the training data is currently usually unary information on items being viewed, clicked, purchased, etc, by the respective users, while rating prediction mainly uses rating information itself as training data.

Recently, recommender systems have also been applied to technology enhanced learning for recommending learning objects (e.g., papers, books, courses, etc) to the students (learners) (Manouselis *et al.*, 2010). However, in this area, almost all the works are focused on the construction of recommender systems to recommend the learning objects (resources/materials) to the learners, e.g in (Bobadilla *et al.*, 2009; Ghauth & Abdullah, 2010) etc, and the uses of recommender systems for ITS, especially for student modeling (predicting student performance in our case) is still a new topic which we will exploit in this work.

As presented in Figure 1.3, there is a similar mapping between student modeling in an ITS (generally, in E-learning systems) and recommender systems where student, task (material), and performance score (e.g. mark/grade) would become user, item, and rating, respectively.

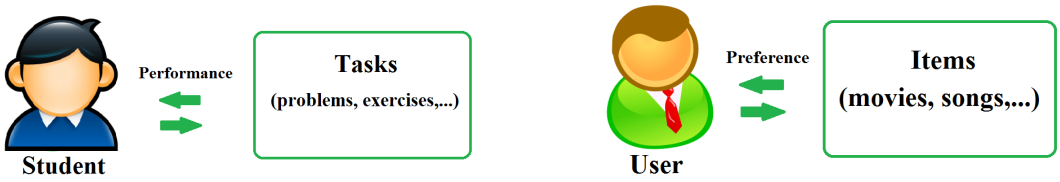


Figure 1.3: "Similarities" between E-learning systems and Recommender Systems

1.1.3 Predicting Student Performance in an ITS

Predicting student performance (PSP), one of the student modeling task in an ITS, has been taken into account recently (Desmarais & Baker, 2011). This task has attracted not only the ITS[1] and the Educational Data Mining[2] communities but also the Machine Learning and Data Mining communities. It was selected as a challenge task for the KDD Challenge 2010[3] at the KDD 2010 Conference and a followed workshop on Knowledge Discovery in Educational Data was also organized at the KDD 2011 Conference.

Specifically, predicting student performance is the task where we would like to know how the students learn (e.g. generally or narrowly), how quickly or slowly they adapt to new problems (Koedinger *et al.*, 2010) or if it is possible to infer the knowledge requirements to solve the problems directly from student performance data (Corbett & Anderson, 1995; Feng *et al.*, 2009), and eventually, we would like to know whether the students perform the tasks (problems/exercises) correctly (or with some levels of certainty) when interacting with the tutoring system.

As discussed in Cen (2009), an improved model for predicting student performance could save millions of hours of students' time and effort in learning Algebra. In that time, students could move to other specific fields of their study or doing other things they enjoy. Moreover, many universities are extremely focused on assessment, thus, the pressure on teaching and learning for examinations leads to a significant amount of time spent for preparing and taking standardized tests. Any move away from standardized and non-personalized tests holds promise for increasing deep learning (Feng *et al.*, 2009). From an educational data mining point of view, an accurate and reliable model in predicting student performance may replace some current standardized tests and thus, reducing the pressure, time, as well as effort on "teaching and learning for examinations" (Feng, 2009). Furthermore, by predicting student performance the instructors could also help the student studying better by providing early feedbacks.

There are several issues in predicting student performance which should be taken into account:

1. The probability that a student guesses correctly while not knowing how to solve the problem at hand or not having the required skills related to the problem, which is called "guess"; and the probability that a student fails (makes a mistake) while knowing how to solve the problem or having all of the required skills related to the problem, which is called "slip" (Corbett & Anderson, 1995).

2. The increase in knowledge over time obviously has an effect on a student's performance, e.g., the second time a student does his/her exercises, his/her performance gets better on average. Furthermore, as an educational factor, the more the students study the better the performance they get. Therefore, the sequential/temporal effect is important information for predicting student performance.

[1] http://en.wikipedia.org/wiki/Intelligent_tutoring_system
[2] http://educationaldatamining.org
[3] https://pslcdatashop.web.cmu.edu/KDDCup/

3. The student effect/bias and task effect/bias, which represent how good/clever a student is and how difficult/easy a task is.

4. The correlations between the students and between the tasks. For example, the students who have similar performances on the past problems may also have similar performances on the future problems.

5. The multiple relationships between students, tasks, and their meta data. For example, each student may perform several tasks, and each task requires one or several skills, while a student is also required in mastering some specific skills.

6. The multiple interactions between the students and the tasks. For example, each student may perform a task several times.

7. The class imbalance problem exists in student performance data. For example, when considering predicting student performance as a binary classification problem, the number of correct solutions are higher than the number of incorrect solutions, thus, causing skew in the target distributions, which may have negative affect on the prediction results.

Taking these issues in mind, this thesis introduces many methods, which mainly base on recommender system techniques, for improving the prediction results in predicting student performance, as summarized in the following section.

1.2 Contribution

The main goal of this thesis is to introduce the methods for predicting student performance. Although some of the methods, for example, matrix and biased matrix factorization, are already existing in recommender systems (e-commerce domain), using them for student modeling (in e-learning domain), especially for predicting student performance, is still a new approach. Our main contributions are to propose using the latent factor models for student modeling; to introduce personalized forecasting as well as tensor factorization methods for modeling sequential/temporal effects in student's knowledge acquisition progress; to exploit multi-relational characteristics in student performance data; to open a new issue in recommendation for e-learning: student's task recommendation; and to deal with class imbalance problem in student performance data (generally, in other machine learning applications).

Thus, in this thesis we bring together several different domains such as intelligent tutoring systems, recommender systems, (educational) data mining, machine learning, personalization, and forecasting. Specifically, our contributions are summarized as follows:

- **Predicting student performance formulation**. We introduce several main characteristics of student performance data, which may have high impact on the

prediction results. For example, those characteristics are student effect, task effect, sequential/temporal effect, multi-relationships, and imbalanced data. We formulate the problem of predicting student performance. We then show how to map the predicting student performance to rating prediction task in recommender systems, to forecasting problem, and to regression/classification problem. These works were published in parts of (Thai-Nghe *et al.*, 2010c, 2011a,g).

- **Student modeling with latent factor models**. We propose using matrix factorization and biased matrix factorization models to implicitly take into account the student and task latent factors (e.g., slip and guess). Moreover, we also encode the "student effect/bias" (e.g., how good/clever a student is, in performing the tasks) and "task effect/bias" (e.g., how difficult/easy a task is) into the models. Since "similar students" may have "similar performances", we propose taking into account the correlations between the students and the tasks by using user/item k-nearest neighbors collaborative filtering. Parts of these works were published in (Thai-Nghe *et al.*, 2010c, 2011b).

- **Exploiting multi-relational aspects in student modeling**. In predicting student performance, each student performs several tasks, and each task requires one or many skills, while the students are also required to master the skills that they have learned. Thus, we propose to exploit such multiple relationships by using multi-relational matrix factorization (MRMF) to improve the prediction results. We noticed that the main relation, which contains the target variable, is more important than the complementary relations. Therefore, we also propose a weighted multi-relational matrix factorization (WMRMF) to take into account the main relation, e.g. getting higher weight than the others. Besides using for predicting student performance, the WMRMF can also be used for other relational domains such as "music recommendation" or "link prediction". These contributions were presented in (Thai-Nghe *et al.*, 2011c).

- **Personalized forecasting**. Since the student performance (or student knowledge) cumulates and improves over time, there is a "trend line" in student's performance. Similar to other domains, where the target distribution shows the "trend line", forecasting techniques are suitable choices. Furthermore, there is a natural fact that "if student A is clever (having higher performance) than student B then we should not use B's performance to predict A's". To take into account these issues, we propose the personalized forecasting methods which plug individualization into the forecasting models. In this approach, instead of using all historical data to form the models as other methods in the literature, the proposed methods only use information of individual student for forecasting his/her own performance. We published these contributions in (Thai-Nghe *et al.*, 2011g).

- **Modeling temporal effects in student knowledge acquisition**. From educational point of view, the students (or generally, the learners) would improve

their knowledge over time, e.g., as mentioned before, the second time a student does his/her exercises, his/her performance gets better on average. Thus, temporal/sequential information obviously has an effect on the student performance and it is an important factor for predicting student performance. Also, inspired from the latent factor models and personalized forecasting models, we propose tensor factorization and tensor factorization forecasting methods to model both the student/task latent factors and the sequential/temporal effects. These works were published in (Thai-Nghe *et al.*, 2011a,f).

- **Student's task recommendations**. We open an issue for student's task recommendation that can tackle the problem in the literature, which is "the preferred learning activities of students might pedagogically not be the most adequate". In the proposed method we can recommend the tasks to the student using "student performance" instead of "student preference". With this approach, we can recommend the suitable tasks to students by filtering out the tasks that are too easy (high predicted performance) or too hard (low predicted performance), or both, depending on the system goal. Furthermore, we propose using context-aware approach for student's task recommendation. Contradictory to recommender systems in e-commerce area where each user is assumed to rate for an item once, in e-learning environment each student may perform a task several times. Thus, using the context-aware methods to utilize multiple interactions of student-task pairs is potentially important for the predictions, and eventually for task recommendations. These contributions were appeared in (Thai-Nghe *et al.*, 2011e).

- **Learning from imbalanced data**. As mentioned before, student performance data are usually skewed (i.e., the number of correct solutions are usually higher than the number of incorrect solutions). This phenomenon is one of the problems that hinder many machine learning classifiers. We introduce several methods for dealing with imbalanced data. These methods belong to the combination approach such as combining cost-sensitive learning or thresholding with sampling methods. They can be used not only for educational data but also for other areas (e.g. network intrusion detection, fraud detection, etc) where the data sets are skewed in their class distributions. Furthermore, we also propose a new evaluation measure for evaluating the classifiers when learning from imbalanced data. These works were presented in (Thai-Nghe *et al.*, 2009, 2010a,b,d, 2011d). However, among these publications, only the article Thai-Nghe *et al.* (2009) was evaluated using educational data from an academic system, and the others were evaluated using data sets from other applications of machine learning.

- **Empirical evaluation and analysis**. We evaluate the proposed methods using both small and large publicly available data sets. We have compared the propose methods with other state-of-the-art methods in both student modeling and recommender system domains. We have also compared the proposed methods for the class imbalance problem with the state-of-the-art methods in that domain. We

empirically show that, in most of the cases, the proposed methods can improve the prediction results compared to the others. Thus, these results validate that the proposed approach could be a good choice for student modeling, especially predicting student performance.

1.3 Published Work

The contributions of this thesis were published in several international conferences, workshops, and a book chapter. The list of publications is as follows:

- Thai-Nghe, N., Drumond, L., Horváth, T., , Nanopoulos, A. & Schmidt-Thieme, L. (2011a). *Matrix and tensor factorization for predicting student performance.* In Proceedings of the International Conference on Computer Supported Education (CSEDU 2011), 69-78. SciTePress Digital Library.
  Best Student Paper Award.

- Thai-Nghe, N., Drumond, L., Horváth, T., Krohn-Grimberghe, A., Nanopoulos, A. & Schmidt-Thieme, L. (2011b). *Factorization techniques for predicting student performance.* Chapter 6 in Educational Recommender Systems and Technologies: Practices and Challenges (ERSAT 2011), in O.C. Santos & J.G. Boticario (eds.), IGI Global (In press).

- Thai-Nghe, N., Drumond, L., Horváth, T. & Schmidt-Thieme, L. (2011c). *Multi-relational factorization models for predicting student performance.* In Proceedings of the KDD 2011 Workshop on Knowledge Discovery in Educational Data (KDDinED 2011). Held as part of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining.

- Thai-Nghe, N., Gantner, Z. & Schmidt-Thieme, L. (2011d). *A new evaluation measure for learning from imbalanced data.* In Proceeding of the IEEE International Joint Conference on Neural Networks (IJCNN 2011), IEEE Xplore.
  Student Travel Grant Award.

- Thai-Nghe, N., Horváth, T. & Schmidt-Thieme, L. (2011e). *Context-aware factorization models for student's task recommendation.* In Proceedings of the UMAP 2011 International Workshop on Personalization Approaches in Learning Environments (PALE 2011). CEUR-WS (ISSN 1613-0073).

- Thai-Nghe, N., Horváth, T. & Schmidt-Thieme, L. (2011f). *Factorization models for forecasting student performance.* In Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and Stamper, J. (Eds.): Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011).

- Thai-Nghe, N., Horváth, T. & Schmidt-Thieme, L. (2011g). *Personalized forecasting student performance.* In Proceedings of the 11th IEEE International Conference on Advanced Learning Technologies (ICALT 2011), IEEE Computer Society.

- Thai-Nghe, N., Do, T.N. & Schmidt-Thieme, L. (2010a). *Learning optimal threshold for Bayesian posterior probabilities to mitigate class imbalance problem.* In Proceedings of the 3rd International Conference on Theories and Applications of Computer Science (ICTACS 2010), 38 - 49.

- Thai-Nghe, N., Do, T.N. & Schmidt-Thieme, L. (2010b). *Learning optimal threshold on resampling data to deal with class imbalance.* In Proceedings of the IEEE International Conference on Computing and Telecommunication Technologies (RIVF 2010), IEEE Xplore.

- Thai-Nghe, N., Drumond, L., Krohn-Grimberghe, A. & Schmidt-Thieme, L. (2010c). *Recommender system for predicting student performance.* In Proceedings of the ACM RecSys 2010 Workshop on Recommender Systems for Technology Enhanced Learning (RecSysTEL 2010), volume 1, issue 2, 2811 - 2819, Elsevier's Procedia Computer Science.

- Thai-Nghe, N., Gantner, Z. & Schmidt-Thieme, L. (2010d). *Cost-sensitive learning methods for imbalanced data.* In Proceeding of the IEEE International Joint Conference on Neural Networks (IJCNN 2010), 1-8, IEEE Xplore.
  Student Travel Grant Award.

- Thai-Nghe, N., Busche, A. & Schmidt-Thieme, L. (2009). *Improving academic performance prediction by dealing with class imbalance.* In Proceedings of the 9th IEEE International Conference on Intelligent Systems Design and Applications (ISDA 2009), 878 - 883, IEEE Computer Society.

1.4 Chapter Overview

The thesis is organized as follows:

- In Chapter 2 we formulate the problem of predicting student performance

- In Chapter 3 we describe data sets that are used for experiments. We show how to map the student performance data to user-item-rating in recommender systems and to forecasting / regression / classification problems

- Chapter 4 summarizes the state-of-the-art methods and the related work in student performance prediction

- Chapter 5 carefully presents the use of latent factor models for predicting student performance. We present step-by-step how the models work, how to implement them and how to use them to make prediction. We also describe how to use k-nearest neighbors collaborative filtering for taking into account the correlations between the students and the tasks
- In Chapter 6 we show how to exploit the multiple relationships in student performance data
- Chapter 7 provides the personalized forecasting methods for taking into account the sequential/temporal effects in predicting student performance
- Chapter 8 is an inspiration from Chapters 5 and 7, in which we will propose the tensor factorization methods for modeling the temporal effect as well as incorporating the forecasting methods into the latent factor models
- Chapter 9 introduces the student's task recommendation. This chapter also proposes exploiting multiple interactions between the students and the tasks by using context-aware factorization methods
- Chapter 10 introduces several methods as well as a new metric for learning from imbalanced data. These methods can apply not only for student performance data but also for other applications of machine learning, such as network intrusion detection, customer retention detection, etc.
- Finally, Chapter 11 puts all the proposed methods into context for comparison and conclusion. We also give an outlook in this area and raise some works for the future

Chapter 2

Problem Definition

Contents

2.1 Problem Formulation 12
2.2 Examples of Predicting Student Performance 15
2.2.1 Predicting Student Performance in an ITS 16
2.2.2 Predicting Student Performance in an Academic System . . . 17

2.1 Problem Formulation

Traditionally, Intelligent Tutoring Systems (ITSs) allow the students solving the problems (exercises) with a graphical front-end that can automate some tedious tasks, provide some hints and provide feedbacks to the students. Such systems can profit from anticipating student performance in many ways, e.g., in selecting the right mix of exercises, choosing an appropriate level of difficulty and deciding about possible interventions such as hints. Thus, the ITSs are valuable environments for collecting data for prediction and for interacting with students in an intelligent way (see more, e.g., in (Massey *et al.*, 1988; Woolf, 2008)).

The problem of student performance prediction in the ITSs is to predict the likely performance of a student for some given problems, or exercises, or part thereof such as for some particular steps, which we call the *tasks*. The task could be to solve a particular *step* in a *problem* (aka, problem-step), to solve a whole problem or to solve problems in a *section* or *unit*, etc. In ITS, the tasks usually are described in two different ways:

- First, the tasks can be located in a topic hierarchy, for example

$$unit \supseteq section \supseteq problem \supseteq step$$

- Second, the tasks can be described by additional meta data such as task descriptions or *skills* that are required to solve the tasks (note that in some systems, the skill is also called *knowledge component*, e.g. in Koedinger *et al.* (2010))

$$skill_1, skill_2, \ldots, skill_n$$

All of these information, the topic hierarchy, the skills and other task meta data can be described as attributes of the tasks. In the same way, the attributes about the students may also be available (e.g. student descriptions, student demographic information, etc).

As already presented in Figure 1.2, the Student Model is used as the input to the Tutoring Module (Beck *et al.*, 1996). Thus, the ITS allows to collect a rich amount of information about how a student interacts with the tutoring system and about his past successes and failures. Usually, such information is collected in a click-stream log with an entry for every action the student has taken. The click-stream log may contain many useful information, e.g. about the

$$time, student, context, action$$

For student performance prediction, such click streams can be aggregated to the task for which the performance should be predicted and eventually be enriched with additional information. For example, if the aim is to predict the performance for each single step in a problem, then all actions in the click-stream log belonging to the same student and step will be aggregated to a single transaction and enriched, for example, with some performance scores (marks / grades).

More formally, let S be a set of students (note that because of privacy concerns, the set we usually have is the student IDs), I be a set of tasks, and $P \subseteq \mathbb{R}$ be a range of possible performance scores (e.g., $P \in [0..1]$). Let M_S be a set of student meta data descriptions and $m_S : S \to M_S$ be the meta data for each student. Let M_I be a set of task meta data descriptions and $m_I : I \to M_I$ be the meta data for each task.

Finally, let

$$\mathcal{D}^{train} \subseteq S \times I \times P$$

be observed student performances and

$$\mathcal{D}^{test} \subseteq S \times I \times P$$

be unobserved[1] student performances.

Then the problem of predicting student performance is, given $\mathcal{D}^{train}$ (in certain cases, also given m_S and m_I) to find

$$\hat{p} : S \times I \to \mathbb{R}$$

[1]Please note that for model testing purpose, P in $\mathcal{D}^{test}$ is know, however, it is usually "hidden"; and for future prediction, P is obviously unknown

such that the measure $\mathcal{E}(\hat{p}, p)$ satisfies a certain condition, e.g., $\mathcal{E}$ needs to be minimum, where $\hat{p}$ is a predicted performance and p is the true performance determined on $\mathcal{D}^{test}$

$$p : S \times I \to P, \qquad (s, i) \mapsto p, \qquad (s, i, p) \in \mathcal{D}^{test}$$

For example, in case of the measure $\mathcal{E}$ is the Root Mean Squared Error (RMSE), $\mathcal{E}$ is determined by

$$\mathcal{E} = \sqrt{\frac{1}{|\mathcal{D}^{test}|} \sum_{(s,i,p) \in \mathcal{D}^{test}} (p - \hat{p}_{(s,i)})^2} \tag{2.1}$$

and in case of using Mean Absolute Error (MAE) as a measure, $\mathcal{E}$ is determined by

$$\mathcal{E} = \frac{1}{|\mathcal{D}^{test}|} \sum_{(s,i,p) \in \mathcal{D}^{test}} |p - \hat{p}_{(s,i)}| \tag{2.2}$$

Some other error measure could also be considered, though. In principle, this can be considered as a *regression problem*.

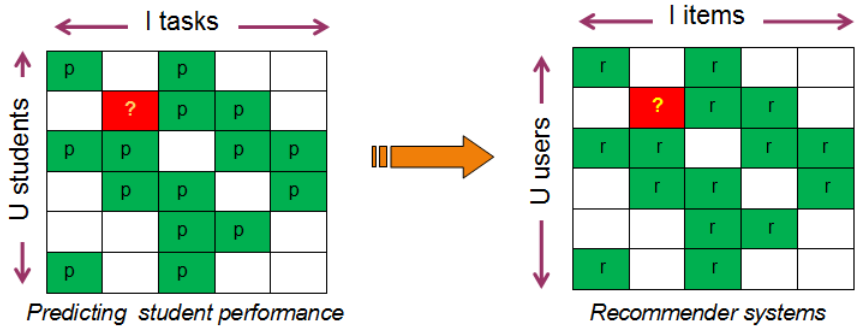


Figure 2.1: Mapping predicting student performance as rating prediction in Recommender Systems (p: performance; r: rating)

For predicting student performance, because of the aforementioned privacy concerns and it is also expensive to gather the student's and task's meta data, we usually have the ID values, e.g. student IDs, task IDs, etc, in the data. Thus, we should find methods which can deal with these ID variables, in this case student ID (s) and task ID (i), both being nominal with many levels (1,000-100,000s). This problem is exactly what the *rating prediction task* in recommender systems does, since student s, task i and performance p would be *user*, *item* and *rating*, respectively. Figure 2.1 presents an example of mapping predicting student performance problem as rating prediction task in recommender systems. Thus, after the mapping, we can apply any recommender system technique to solve this problem.

Furthermore, an important issue as already mentioned in Chapter 1 is the *potentially sequential effect*, which describes how the students gain experience over time. Since the student performance improves over time, there may have a "trend line" in the student performance. We therefore propose considering the predicting student performance as a *forecasting problem*, which is illustrated in Figure 2.2. For example, we can use the past solved problems (steps) to forecast the next ones (we will describe this Figure more details in Chapter 7).

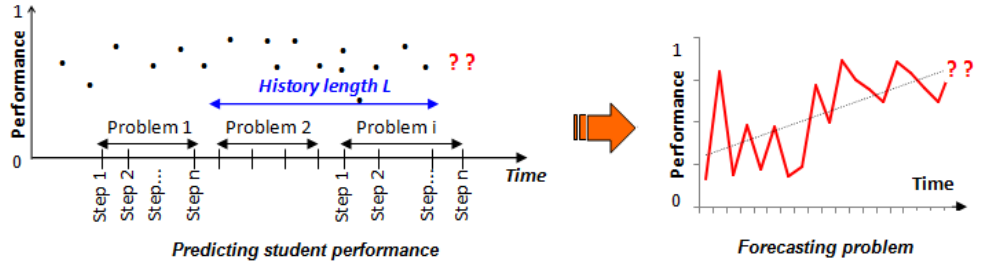


Figure 2.2: Mapping predicting student performance as forecasting problem

Moreover, depending on which measure ($\mathcal{E}$) we are using, the predicting student performance problem can also be considered as a *classification problem*. For example, if one uses the area under the ROC curve (AUC) or F1-Measure as a measure, $\mathcal{E}$ needs to be maximized, instead. By doing so, one can apply any classification techniques, e.g. Decision Tree, Support Vector Machines, etc to solve it. Also, depending on the performance scores used in a specific ITS, this classification problem could be a *multi-class* classification problem (e.g., the student performance is evaluated by ranking [A/ B/ C/...] or [Excellent/ Very Good/ Good/...], etc), or *binary-class* classification (correct/incorrect or 0/1, etc) as in this thesis. However, when we considered the predicting student performance as a binary classification problem, we have discovered an important issue, which may affect to the prediction results, that is the *class imbalance problem* (we will discuss in Chapter 10).

Please note that, in this work, we have formulated the *predicting student performance in an ITS* as *rating prediction* in recommender systems (or forecasting / regression / classification problems), however, we can apply this idea for any other systems by the same way. For example, in academic systems, e.g. Student Information Systems, we could also predict/forecast the student GPA given a specific course (as an item) or predict student CGPA (cumulative GPA) given a specific field of study (as an item) for each term/semester/year, etc, as in (Thai-Nghe, 2006; Thai-Nghe *et al.*, 2007).

2.2 Examples of Predicting Student Performance

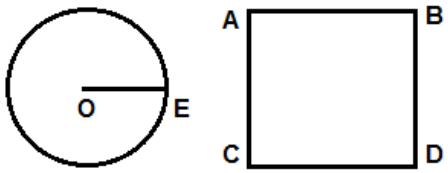
We will provide two typical examples to show that the proposed approach can be used for predicting student performance not only in an ITS, but also in an academic system.

2.2.1 Predicting Student Performance in an ITS

Figure 2.3a presents an example of predicting student performance in an ITS. Given the circle and the square as in this figure, the task for the student could be "What is the remaining area of the square after removing an area which is equal to a circular area?" (Koedinger *et al.*, 2010) To solve this task (question), students could do some smaller subtasks which are called "steps". Each step may require one or more skills (also called "knowledge components"), for example:

step 1) Calculate the circle area (skill: $area_1 = \pi * (OE)^2$)
step 2) Calculate the square area (skill: $area_2 = (AB)^2$)
step 3) Calculate the remaining (skill: $area_2 - area_1$)

Each step is recorded as a transaction. Figure 2.3b presents a snapshot of the transactions. Based on the student's past performance (of the solved tasks), we would like to predict the students' next performance (e.g. correct/incorrect) for some given new tasks.



a. A task example

		Task		Skill		
Row	Student	Problem	Step	Knowledge component	Opportunity Count	Correct at First Attempt
1	S01	WATERING_VEGGIES	(WATERED-AREA Q1)	Circle-Area	1	1
2	S01	WATERING_VEGGIES	(TOTAL-GARDEN Q1)	Rectangle-Area	1	1
3	S01	WATERING_VEGGIES	(UNWATERED-ARE Q1)	Compose-Areas	1	1
4	S01	WATERING_VEGGIES	DONE	Determine-Done	1	0
5	S01	MAKING-CANS	(POG-RADIUS Q1)	Enter-Given	1	?
6	S01	MAKING-CANS	(SQUARE-BASE Q1	Enter-Given	2	?
7	S01	MAKING-CANS	(SQUARE-AREA Q1	Square-Area	1	?
8	S01	MAKING-CANS	(POG-AREA Q1)	Circle-Area	2	?

Past perform-ance

Predict

b. A snaphot of the transaction log

Figure 2.3: An example of predicting student performance in an ITS (picture is adapted from https://pslcdatashop.web.cmu.edu/KDDCup).

2.2.2 Predicting Student Performance in an Academic System

The proposed approach can also be used for predicting student performance in an academic system. For example, many universities are currently using credit systems, and each semester (term) a student has to study some required courses and several elective courses. Imaging that a certain university has ≈40,000 students[1], and each student has to select 4 among 10 elective courses. Almost the students are quite confused on those elective courses since they do not know which ones are suitable for them (even, they do not have any background knowledge about these courses). This problem, thus, requires heavy helps from the teachers/supervisors/advisors (even in some cases, the advisors are also confused since they do not have much background knowledge about their students). Therefore, both the teachers and the students have spent lots of times for doing this task.

With our approach, using student performance on the past (learned) courses, we can easily predict the student performance on the given unlearned courses, for example, by casting student as the *user* and course as the *item*, as presented in Figure 2.4. Based on the prediction results, the students may know, at least, some information about their (predicted) performances on those courses, and may determine which ones are appropriate for their abilities. Also, based on predicting student performance, we can provide them early feedbacks, more tutorials, etc, thus, we can prevent the students dropping (or even expelling) every year (as shown in Thai-Nghe *et al.* (2009), about two hundreds students were expelled at CTU every year, due to poor performance).

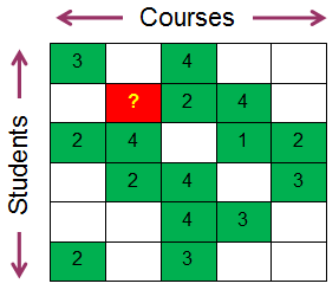


Figure 2.4: An example of predicting student performance in an Academic System

[1] e.g., the Can Tho University (CTU), Vietnam, http://www.ctu.edu.vn

Chapter 3

Data sets and Pre-Processing

Contents

3.1 Data Sets **18**
3.1.1 Cognitive Tutor Data (KDD Cup 2010 Data) 19
3.1.2 ASSISTments Platform Data 20
3.2 Data Mapping for Collaborative Filtering **21**
3.3 Data Mapping for Regression Problem **24**
3.4 Data Mapping for Classification Problem **24**
3.5 Data Encoding and Dealing with Multiple Skills **25**
3.6 Data Splitting Protocol **26**

In this Chapter, we first describe the original data sets that we have selected for experiments. We then show how to map these data into the context of *user*, *item* and *rating* in recommender systems which are used for the proposed techniques, e.g. factorization models, personalized forecasting, etc. Finally, we also present how to map these data into regression problem which can be used by traditional regression techniques such as logistic regression.

3.1 Data Sets

For testing the proposed approached, we use three published data sets which are extracted from the Cognitive Tutor (used for the Knowledge Discovery and Data Mining Challenge 2010 - KDD Cup 2010[1]) (Koedinger *et al.*, 2010) and from the ASSISTments Platform[2] (Feng *et al.*, 2009).

[1] https://pslcdatashop.web.cmu.edu/KDDCup/
[2] http://teacherwiki.assistment.org/wiki/Assistments_2009-2010_Full_Dataset

These data sets, originally labeled "*Algebra 2008-2009*", "*Bridge to Algebra 2008-2009*" and "*ASSISTments-2009-2010*", will be denoted "**Algebra**", "**Bridge**" and **ASSISTments**, respectively, for the remainder of our work. The Algebra and Bridge were already splitted into train and test partitions. These data sets are quite large and their original information are described as in Table 3.1.

Table 3.1: Information of Original Data Sets

Data set	Size	#Attributes	#Instances
Algebra 2008-2009 train	3.1 GB	23	8,918,054
Algebra 2008-2009 test	124 MB	23	508,912
Bridge to Algebra 2008-2009 train	5.5 GB	21	20,012,498
Bridge to Algebra 2008-2009 test	135 MB	21	756,386
ASSISTments-2009-2010	118 MB	20	1,011,079

These data sets represent the log files of interactions between the students and the tutoring systems. While the students solve the problems (e.g math, geometric, etc), their activities, success and progress indicators are logged as individual rows in the data sets.

3.1.1 Cognitive Tutor Data (KDD Cup 2010 Data)

In the Algebra and Bridge data sets, the central element of interaction between the students and the tutoring system is the *problem* (aka, problem name). Every problem belongs into a *hierarchy* (aka, problem hierarchy) of *unit* and *section*. Furthermore, a problem consists of many individual *steps* (aka, step name) such as calculating a circle's area, solving a given equation, entering the result and alike. Generally, students are not required to solve the steps of a given problem in a special order and, as logged by the field *problem view*, which tracks how many times the student already saw this problem.

Additionally, for both Algebra and Bridge data sets, a different number of *knowledge components* (KC, aka, *skills*) and associated *opportunity counts* are provided. The knowledge components represent specific skills used for solving the problem (where available) and the opportunity counts encode the number of times the respective knowledge component has been encountered before.

In this work, we use the KC-Rules column (as the skill) on the Algebra data set. Unfortunately, *the KC-Rules column is not provided on the Bridge data set, and we have used the KC-TracedSkills column instead.* According to the descriptions on the KDD Challenge 2010 Website, the KC-Rules column has more informative than the KC-TracedSkills column since it is a more fine-grained categorization of similar steps (the KC-TracedSkills column is a more coarse-grained categorization). Moreover, the

KC-Rules may provide clues to a better clustering of steps to predict transfer of learning (Koedinger *et al.*, 2010).

Furthermore, in the training data but not in the test data, *incorrects*, *corrects*, and *hints* track the number of failed or successful attempts on the respective step and the number of times the student requested additional information from the tutoring system. Furthermore, fields such as time information (first transaction time, correct transaction time, ...) are given for the training sets but not for the test sets.

Target of the prediction task in both Algebra and Bridge is the *correct first attempt* (CFA) information which encodes whether the student successfully completed the given step on the first attempt. This target CFA is encoded by 0 (indicating incorrect answers) or 1 (indicating correct answers). Thus, the prediction could be considered as the *certainty* that the student will succeed on the first try.

A snapshot of these two data sets, which covers the attributes that we have used for experiments, is presented in Figure 3.1. For more information, please see Koedinger *et al.* (2010) and descriptions from the KDD Cup 2010 website[1].

Student ID	Problem Hierarchy	Problem Name	Problem View	Step Name	Knowledge Components	Correct First Attempt
stu_de2777346	Unit CTA1_01, Section CT/	L1FB12	1	R1C1		1
stu_de2777346	Unit CTA1_01, Section CT/	L1FB12	1	R1C2		0
stu_de2777346	Unit CTA1_01, Section CT/	L1FB12	1	R2C1		1
stu_de2777346	Unit CTA1_01, Section CT/	L1FB12	1	R2C2	Identifying units	1
stu_de2777346	Unit CTA1_01, Section CT/	L1FB12	1	R3C1	Define Variable	1
stu_de2777346	Unit CTA1_01, Section CT/	L1FB12	1	R3C2	Write expressior	1
stu_de2777346	Unit UNIT-CONVERSIONS	UNITCONVEF	1	MoreOrFew	Compare units	0
stu_de2777346	Unit UNIT-CONVERSIONS	UNITCONVEF	1	ConversionI	Enter unit conve	1
stu_de2777346	Unit UNIT-CONVERSIONS	UNITCONVEF	1	SelectFracti	Select form of or	1

Figure 3.1: A snapshot of the KDD Cup 2010 data sets

3.1.2 ASSISTments Platform Data

The ASSISTments data set is published by the ASSISTments Platform (Feng *et al.*, 2009) that allows teachers to write individual ASSISTments. Each ASSISTments[2] is composed of questions and associated hints, solutions, web-based videos, etc.

The structure of this data set is quite similar to the KDD Cup 2010 data sets. Instead of having the hierarchy as in the KDD Cup 2010 data

$$ProblemHierarchy\ (unit, section) \supseteq ProblemName \supseteq StepName$$

[1] https://pslcdatashop.web.cmu.edu/KDDCup

[2] The word "ASSISTment" blends tutoring "assistance" with "assessment" reporting to teachers (http://teacherwiki.assistment.org/wiki/About)

the ASSISTments data set has the structure

$$Assignment \supseteq Assistment \supseteq Problem$$

From this structure, we can easily recognize that each Assignment has one or many ASSISTments, and each ASSISTment consists of one or more problems.

Moreover, instead of naming "knowledge component" as in the KDD Cup 2010 data, the ASSISTments calls it the *skill*. Target of the prediction task is also the *correct first attempt* information which encodes whether the student successfully completed the given *problem* on the first attempt.

A snapshot of this data set, which includes the attributes that we have used for experiments, is presented in Figure 3.2. For more information, please see Feng *et al.* (2009) and descriptions from the ASSISTments Platform[1].

user_id	assignment_id	assistment_id	problem_id	list_skills	correct
73963	232368	42904	76429		0
73963	232368	42904	76430		1
78068	259873	55546	97478		1
78068	253099	32044	49028	Mean;Table	0
78068	253099	32044	49029		1
78068	253099	32044	49030		0
63205	260543	40113	68274		1
63205	260543	40113	68275	Multiplication and Divis	1
70759	232359	42034	74567	Probability Compound	0

Figure 3.2: A snapshot of the ASSISTments data set

Except the other attributes in these data sets which we have not used yet, the only difference between the KDD Cup 2010 and the ASSISTments data is just the naming convention (please note that in the KDD Cup 2010 data, the task is to predict a single step while the task is to predict a single problem in the ASSISTments data).

Therefore, for easiness in naming convention reason, we assume that these data share some similarities (although they come from different platforms and may aim at different purposes). Using our notations, we denote them in a uniform way, as presented in Table 3.2

3.2 Data Mapping for Collaborative Filtering

In traditional recommender system settings, it is unambiguous how the available information is mapped to users, items, and ratings, respectively. At least for all major

[1] http://teacherwiki.assistment.org

Table 3.2: Data sets' attributes in our notations

Our notation	Algebra & Bridge Data	ASSISTments Data
S	Student	User
I_1	Problem Hierarchy (Unit + Section)	Assignment
I_2	Problem Name	Assistments
I_3	Step Name	Problem
I_4	Problem View	-
I_5	Problem Group (the first part of I_2)	-
I_6	Knowledge component	Skill
I_7	$I_1 + I_2 + I_3 + I_4$	$I_1 + I_2 + I_3$
P	"Correct first attempt"	"Correct first attempt"
Prediction task	a single step	a single problem

recommender system data sets used (e.g., Jester[1], Movielens 100k[2], and Netflix[3]) there is a unique assignment[4].

For predicting student performance by using recommender system techniques, an important problem is that which *task* will be considered as *item*. There is an obvious mapping of users and ratings given the Algebra, Bridge and ASSISTments data sets:

$$\begin{gathered} \textit{student} \;\mapsto\; \textit{user} \\ \textit{correct first attempt (performance)} \;\mapsto\; \textit{rating} \\ \textit{?} \;\mapsto\; \textit{item} \end{gathered}$$

For mapping the *item*, several options seem to be available to us. From the data set description it was immanent that an *item* was supposed to be the combination (concatenation) of I_1, I_2 and I_3 (refer to Table 3.2 for these notations). For example, in Algebra and Bridge data sets, choosing I_7 as an *item* had the drawback of incurring the new-item problem[5] into our recommendation task: in the test sets, instances of I_7 would occur that are unavailable in the training set, thus our models would not be able to learn much about them. For example, in case of the Algebra data set, 24,735 unique step names are not present in train out of a total of 44,730 unique instances; luckily, this affects only 26,378 out of the 508,912 rows in test. Another problem with this approach is that it leads to huge sparsity in the data. For instance, for the Algebra data set this configuration would lead to a total of 1,416,473 items (see Table 3.3). Since there are

[1] http://eigentaste.berkeley.edu/dataset

[2] http://www.grouplens.org/node/73

[3] http://www.netflix.com

[4] The treatment of users and items usually is symmetric and bears only performance implications.

[5] This is a cold-start problem in recommender systems (Schein *et al.*, 2002). We will discuss how to deal with it in Section 5.5.2

8,918,054 examples on the training set for this data set one could expect to see, on average, 6 observations per item (99.81% sparsity). To cope with these problems, we considered ignoring the I_4 component and started thinking about other possibly valid combinations we could use as an *item*. However, this choice may lead to the ambiguity in the prediction since each single step is distinguished by the I_7.

Table 3.3: Mapping student performance data to User-Item-Rating

	Algebra	Bridge	ASSISTments
User	**#User**	**#User**	**#User**
Student	3,310	6,043	8,519
Item	**#Item**	**#Item**	**#Item**
I_7 (solving-step) (*)	1,416,473	887,740	220,524
I_6 (skill) (*)	2,979	1,458	348
$I_1 + I_2 + I_3$	1,309,038	593,369	220,524
$I_1 + I_5 + I_3$	848,218	188,001	-
$I_5 + I_3$	776,155	155,808	-
$I_2 + I_3$	1,254,897	566,843	35,978
I_3	695,674	126,560	35,978
I_2	188,368	52,754	22,039
I_5	185,918	52,189	-
$I_1 + I_2$	206,596	61,848	159,852
$I_1 + I_5$	1,000	1,343	-
I_1	165	186	6,163
$I_1 + I_2 + I_4$	220,045	101,707	-
$I_1 + I_5 + I_4$	3,203	5,537	-
$I_1 + I_5$	780	1,526	-
Rating	**#Rating**	**#Rating**	**#Rating**
Correct first attempt	8,918,054	20,012,498	1,011,079

(*) items are used for the experiments

Generally speaking, there are three important factors for the evaluation of the candidate items:

- the percentage of new users or new items in the test set
- the overall sparsity of the resulting matrix, where

$$\text{sparsity} = 100 - \frac{|P| * 100}{|S| * |I|}$$

- the percentage of missing values

As described in the data sets, the knowledge component (KC-Rules) I_6 is an informative attribute for prediction, however, when choosing any of the knowledge component I_6 as (part of) an *item*, ambiguity is also introduced as there may be more than one knowledge component of a given kind per training row (with a maximum of 17 "rule" type knowledge components being applied to one row in Algebra). Furthermore, the knowledge components had a high degree of missing values with $\approx 20\%$ of the rows not having a knowledge component.

Table 3.3 presents some of the combinations which can be used as the *items*, however, other combinations may also be possible (e.g., $I_1 + I_6$, $I_5 + I_6$, etc). Indeed, as shown later in the experimental results of Chapter 5, different mappings yield different results (we will compare some of these selections and propose using the I_6 or I_7).

3.3 Data Mapping for Regression Problem

As we have already seen, predicting student performance problem can be considered as rating prediction task in recommender systems, however, this problem can also be solved by many traditional approaches such as linear or logistic regressions. Since most of the available columns were categorical in the aforementioned data sets, for employing regression models, we need to pre-process these data before we could regress on them.

A simple way to do is to binarize these categorical attributes. However, this method lead to very large and very sparse data sets, e.g. millions of new binary attributes as shown in (Yu *et al.*, 2010). We provide a simple method, which is mainly derived from the *item average* in recommender systems (Vozalis & Margaritis, 2003), to encode the categorical variables as input for the regression models.

Based on the formulation in Chapter 2, let

$$\mathcal{D}_{s,i} := \{(s', i', p) \in \mathcal{D} \mid s' = s, i' = i\}$$

be the set of transactions logged for student s and task i, then the average value on task i is determined by

$$AVG(i) = \frac{\sum_{(s,i,p)\in\mathcal{D}_{s,i}} p}{|\mathcal{D}_{s,i}|} \tag{3.1}$$

For example, in the data sets from Table 3.3, the variables which we can compute the respective averages are the student ID, solving-step (I_7) and skill (I_6).

3.4 Data Mapping for Classification Problem

In these data sets, since our prediction target variable (the "correct first attempt") is a binary attribute with 1 indicating correct answer and 0 indicating incorrect answer, instead of consider the prediction task as a regression problem in the interval [0..1] as

described above, we may also consider it as a classification task of 0/1 values. Thus, for prediction, one can use other classification methods such as Decision Trees, Support Vector Machines, etc. Depending on which methods we are using, we can keep the categorical attributes, binarize or average them as described in Section 3.3.

However, when we consider the predicting student performance as a (binary) classification problem, an important notice we would like to raise here is the *class imbalance problem*[1], which is known as one of the problems that hinders most of the classifiers (Chawla *et al.*, 2004; He & Garcia, 2009) and also is one of ten challenging problems for machine learning research (Yang & Wu, 2006).

Table 3.4 summarizes the class imbalance information in three data sets. Here, we consider the class 0 (incorrect answer) as the minority class and the class 1 (correct answer) as the majority class.

Table 3.4: Class imbalance information in student performance data

Data set	#Examples	#Minorities	%Minorities	Imbalanced Ratio
Algebra	8,918,054	1,303,324	14.61	5.84
Bridge	20,012,498	2,768,464	13.83	6.23
ASSISTments	1,011,079	375,197	37.11	1.69

3.5 Data Encoding and Dealing with Multiple Skills

In the proposed methods (e.g., matrix/tensor factorization) we will use the ID values, e.g., user (student) ID, item (task) ID, etc. Thus, the data sets should be encoded in term of ID values and all of them should be started from 1 (for computer memory saving reason). To deal with empty values in the attributes of the data sets, we have used a new default ID for them.

Moreover, in student modeling of an ITS, each task can associate with one or more skills. Modeling these multiple skills is a complex and challenging problem within the Cognitive Tutor (Pardos & Heffernan, 2010; Xu & Mostow, 2011). For example, the current state-of-the-art Bayesian Knowledge Tracing can only deal with one skill associated with the task.

To simplify this problem, the multiple skills associated with the task have been collapsed into one single skill. Here, we treated a full string of skills as a single skill (this has been done in Pardos & Heffernan (2010), called "treat a unique set of skills as a completely separate skill"). However, other sophisticated methods could also be considered, e.g., in (Pardos & Heffernan, 2010; Xu & Mostow, 2011).

[1]In binary classification problem, class imbalance is a phenomenon in which the class distribution is far from the uniform distribution. We will discuss and provide methods for dealing with this problem in Chapter 10.

3.6 Data Splitting Protocol

For splitting the data sets into test sets and validation sets, we use the same protocol as described from the KDD Cup 2010[1] which represents in Figure 3.3.

In this figure, each horizontal line represents a transaction between a student and a step (aka, student-step). The data set is broken down by student, unit, section , and problem.

According to the data description, "test rows are determined by a program that randomly selects one problem for each student within a unit, and places all student-step rows for that student and problem in the test file. Based on time, all preceding student-step rows for the unit will be placed in a training file, while all following student-step rows for that unit will be discarded" (Koedinger *et al.*, 2010). This way, temporal information is encoded in the train-test-split.

Figure 3.3: Data splitting protocol (source: KDD Cup 2010)

[1] https://pslcdatashop.web.cmu.edu/KDDCup/rules_data_format.jsp

Chapter 4

State-of-the-art Methods and Related Work

Contents

4.1 Classification and Regression Methods 27
4.2 Bayesian Knowledge Tracing (BKT) 30
4.3 Performance Factors Analysis (PFA) 35
4.4 Selection of the State-of-the-art Methods 36
4.5 Factorization Techniques 38

In this chapter, we will summarize the state-of-the-art methods and the most related works in student modeling, especially in predicting student performance. However, many other works can be found in Baker & Yacef (2009); Desmarais & Baker (2011); Romero & Ventura (2006, 2010); Romero *et al.* (2010).

4.1 Classification and Regression Methods

For predicting student performance outside the tutoring systems, several works have been done by applying classification and regression techniques. Specifically, Bekele & Menzel (2005); Kotsiantis & Pintelas (2005); Minaei-Bidgoli *et al.* (2003) proposed using Bayesian networks, decision trees, and other classification techniques to predict the student results; Delavari *et al.* (2004) proposed a model with different types of education-related questions and data-mining techniques which are appropriate for them. For examples, predicting student performance, clustering similar students, and associating types of students with appropriate courses; Romero *et al.* (2008) compared different data mining techniques (e.g. neural networks, decision trees, etc.) to classify

students based on their Moodle usage data and the final marks obtained in their respective courses. In preliminary works, we have also analyzed and compared several classification methods (e.g. decision trees and Bayesian networks) for predicting academic performance, for example, predicting students' GPA at the end of the first year in their Master course (Thai-Nghe, 2006; Thai-Nghe *et al.*, 2007). We also proposed to improve the student performance prediction by dealing with the class imbalance problem (i.e., the ratio between passing and failing students is usually skewed), which we will thoroughly present in Chapter 10. These works, however, are to predict student performance in general academic systems rather than in the tutoring systems. Usually, these classification methods work well if we have enough background knowledge of the domain (e.g., the meta data about the students).

Recently, for predicting student performance in the tutoring systems at the KDD Challenge 2010, researchers have proposed using feature engineering and ensembling techniques, e.g., in (Shen *et al.*, 2010; Tabandeh & Sami, 2010; Yu *et al.*, 2010).

Precisely, Tabandeh & Sami (2010) proposed their approach by deleting the features which are not presented in the test set and created some new features by transforming the categorical features into numerical ones. Finally, based on the new numeric features, they built a regression tree for final prediction.

In another work, Yu *et al.* (2010), who are the winner of the KDD Challenge 2010, have proposed using feature engineering and ensembling techniques. The feature engineering approaches are categorized into two types: sparse features which are generated by binarization and discretization techniques, and condensed features which are generated by using statistics on the data. We briefly summarize these approaches in the following.

- **Sparse Features**

  There are four groups of sparse features:

  i) Basic Sparse Features are categorized into two types: categorical features (e.g., unit name, section name, etc) and numerical features (e.g., problem view and opportunity count). The categorical features were expanded into a set of binary features, while the numerical feature x was linearly scaled to the range [0..1] by using $x/(x+1)$ or nonlinearly scaled by using $log(1+x)$.

  ii) Feature Combination is used on the pairs of features, e.g., (student name, unit name), (unit name, section name) etc, to generate the new ones. This approach is based on the polynomial mapping in kernel methods or bigram/trigram in natural language processing.

  iii) Temporal Features are used to take into account the temporal effect by embedding temporal information into feature vectors. For example, in each step, the step name and the skill values from the previous few steps were added as features.

  iv) Other Generating Features: Each (string) value of the feature is tokenized as a

binary feature. For example, a skill "Write expression, positive slope" is tokenized in to four features, "Write", "expression", "positive" and "slope". Furthermore, the clustering techniques are also used to group similar problems or steps together. For example, two steps "-18+x = 15" and "5+x = -39" can be considered as the same type of steps. In addition, the authors have also tried to model the learning experience of students since the student performance may depend on whether that student had previously encountered the same step or problem. Thus, other features were added to indicate such information.

Finally, all of the above features are used for training a logistic regression via LIBLINEAR (Fan *et al.*, 2008).

- **Condensed Features**

  Learning Temporal Information: Two approaches have been proposed to extract the temporal information.

  In the first approach, the authors state that the performance on the current problem is related to the same student's past performances on similar types of problems. Thus the average performance and average hint on the student's previous steps (up to six) with the same skill are used to model the student's recent performance on similar problems. A binary feature is added to indicate whether such previous records exist.

  In the second approach, for the current step, the authors proposed to find the skill and check whether the student has seen a step with the same skill within the same day, 1-6 days, 7-30 days, or more than 30 days. Thus, four binary features are used to indicate how familiar a student is with the given skill.

  Correct First Attempt Rates (CFAR)

  The CFAR is used to predict the performances of the students on the item i by averaging the correct performances of that item, up until the current point. More formally, let

  $$\mathcal{D}_i := \{(i', p) \in \mathcal{D}^{train} \mid i' = i\}$$

  be the set of transactions of the item i, up until the current prediction point, then the performance of all students on item i is predicted by

  $$\hat{p}_i = \frac{\sum_{(i,p)\in\mathcal{D}_i \ \wedge \ p=1} p}{|\mathcal{D}_i|} \tag{4.1}$$

  The authors have used the CFAR on several items such as step name, problem name, skill, (problem name, step name), etc.

  These condensed features are finally used for training the Random Forests (Breiman, 2001).

- **Ensembling:** All the above classifiers are finally ensembled by using regularized linear regression.

By using these feature engineering techniques, the authors have come up with millions of features (21,684,170 features on Algebra data set and 30,971,151 features on Bridge data set). Thus, these works require much human effort on data preprocessing as well as requiring intensive-memory of the computers.

4.2 Bayesian Knowledge Tracing (BKT)

The Bayesian Knowledge Tracing model (Corbett & Anderson, 1995) is one of the state-of-the-art methods for student modeling. This method can be used to predict the student performance and to determine whether the student has mastered a particular skill. There are three assumptions in the BKT:

- first, each skill has two states: "learned" or "unlearned";
- second, the skill can make a transition from the unlearned to the learned state at each opportunity to apply the skill;
- third, there is no forgetting, which mean that skills can not make the transition from the learned state back to the unlearned state.

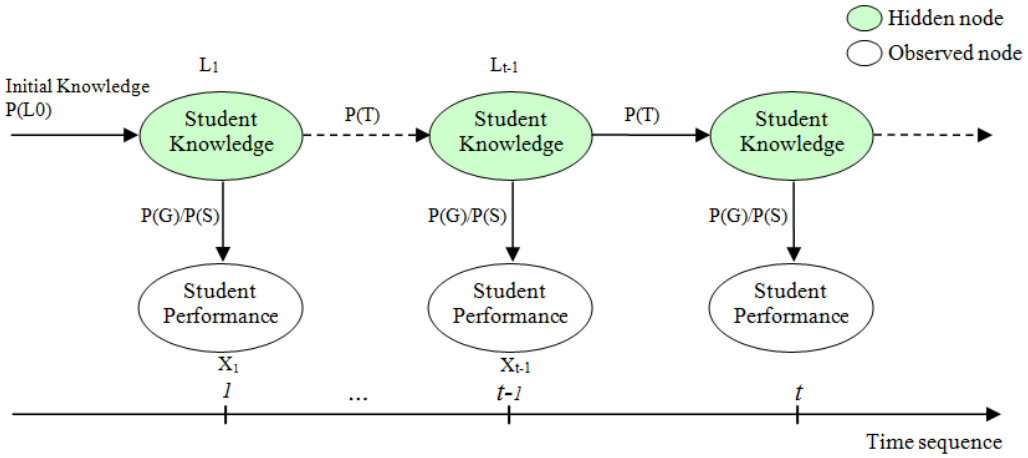


Figure 4.1: Bayesian Knowledge Tracing as an Hidden Markov Model

As presented in Figure 4.1 (adapted from Gong *et al.* (2010a)), the BKT is an Hidden Markov Model (HMM) with a hidden node (student knowledge) and an observed node (student performance). In this model, each skill has four parameters: two knowledge parameters and two performance parameters.

The two knowledge parameters are *initial knowledge* or *prior knowledge*, denoted as $P(L_0)$ (or $P(L_0 = 1)$) and *learn rate* (transition probability), denoted as $P(T)$ (or $P(L_t = 1|L_{t-1} = 0)$). The initial knowledge $P(L_0)$ is the probability that a particular skill was known by the student before interacting with the tutoring system, e.g., the student might already know that skill from their friends, or from reading papers, etc (this parameter also represents for the skill in the learned state at time 0). The learn rate $P(T)$ is the probability that the skill will make transition from the unlearned state to the learned state at each opportunity of applying that skill. Its conditional probability distribution (CPD) can be presented as in Table 4.1.

Table 4.1: The CPD of learn rate: $P(L_t|L_{t-1})$

	$L_{t-1} = 0$	$L_{t-1} = 1$
$L_t = 0$	$1 - P(T)$	0
$L_t = 1$	$P(T)$	1

The two performance parameters are *guess rate* (guess) $P(\mathbf{G})$ and *slip rate* (slip) $P(\mathbf{S})$. The guess $P(\mathbf{G})$ is the probability that a student solves the problem correctly even though he/she does not know the skill (the skill is in the unlearned state). The slip rate $P(\mathbf{S})$ is the probability that a student does not solve the problem correctly (making a mistake) even though he/she knows the required skill (the skill is already in the learned state).

Table 4.2: The CPD of $P(X_t|L_t)$

	$L_t = 0$	$L_t = 1$
$X_t = 0$	$1 - P(\mathbf{G})$	$P(\mathbf{S})$
$X_t = 1$	$P(\mathbf{G})$	$1 - P(\mathbf{S})$

The BKT model has two phases: The first phase is learning four parameters $P(L_0)$, $P(T)$, $P(\mathbf{G})$, and $P(\mathbf{S})$; and the second phase, which is a main part of the BKT, is tracing the student knowledge.

In principle, the BKT's parameters can be learned by using any HMM learning algorithm (dynamic Bayesian networks), e.g., using Baum-Welch Algorithm (Welch, 2003), or generally, using Expectation Maximization (EM) (Murphy, 2001). Since we only focus on the main part of the BKT (the second phase), we will not describe how to learn the HMM or EM in details. For general learning algorithms of the HMM, we refer the advanced readers to other literature, e.g., see (Murphy, 2001; Rabiner & Juang, 1986; Rabiner, 1990; Welch, 2003).

In educational data mining domain, for instance, to learn these four parameters, there are some published works such as applying the aforementioned Expectation Maximization (Chang *et al.*, 2006) which is denoted as **BKT-EM**, using Dirichlet priors

(Beck & Chang, 2007), or just using a Brute-Force (grid search) which is denoted as **BKT-BF** (Baker *et al.*, 2008b). We consider to use the BKT-EM and BKT-BF for comparisons in this thesis since they are the most popular methods in the educational data mining community. We will briefly describe their learning processes in the following:

- The EM is used to determine the BKT's parameters by finding a set of parameters that maximize the data likelihood (the probability of observing the student performance data). It considers the student performance as evidence with time order, and uses this evidence for the expectation step where the expected likelihood is calculated. Then, the model computes the parameters which maximize that expected likelihood. It iteratively runs the E-step and M-step until it finds the final best fitting parameters (Baker *et al.*, 2011; Chang *et al.*, 2006; Gong *et al.*, 2010a).

- The Brute-Force is used to determine the four parameters of the BKT by using a hyper parameter search procedure (using an exhaustive search) which looks for the best hyper parameters in the entire parameter spaces (Baker *et al.*, 2008b). The drawback of this method is that it suffers from a computational cost, especially on large data sets.

Now, suppose that the learning process is finished, the four parameter $P(L_0)$, $P(T)$, $P(\mathbf{G})$, and $P(\mathbf{S})$ are found. We then use them for tracing the student knowledge, and eventually, predicting student performance (this is an inference process in Bayesian methods).

Let X denote the observed node (student performance node), where $X = 1$ and $X = 0$ indicate for correct and incorrect performances, respectively.

Then, the probability that the student has learned/mastered the skill at opportunity (time) t is computed by

$$P(L_t) = P(L_{t-1}|X_t) + (1 - P(L_{t-1}|X_t)) * P(T) \tag{4.2}$$

where

$$P(L_{t-1}|X_t = 1) = \frac{P(L_{t-1}) * (1 - P(\mathbf{S}))}{P(L_{t-1}) * (1 - P(\mathbf{S})) + (1 - P(L_{t-1})) * P(\mathbf{G})} \tag{4.3}$$

and

$$P(L_{t-1}|X_t = 0) = \frac{P(L_{t-1}) * P(\mathbf{S})}{P(L_{t-1}) * P(\mathbf{S}) + (1 - P(L_{t-1})) * (1 - P(\mathbf{G}))} \tag{4.4}$$

The probability that student s will correctly apply a skill at opportunity (time) $t + 1$ in a sequence of problem solving is predicted by

$$\hat{p}_s = P(L_{st}) * (1 - P(\mathbf{S})) + (1 - P(L_{st})) * P(\mathbf{G}) \tag{4.5}$$

Algorithm 1 Using Bayesian Knowledge Tracing (Corbett & Anderson, 1995) for predicting student performance

1: **procedure** BKT($\mathcal{D}^{train}$)
   /* *Learning the parameters* */
2: **for** each skill i in $\mathcal{D}^{Train}$ **do**
3: $\{P_i(L_0), P_i(\mathbf{S}), P_i(\mathbf{G}), P_i(T)\} \leftarrow$ LearningParameters($\mathcal{D}^{train}$)
4: **end for**

   /* *Knowledge tracing* */
5: **for** each skill i in $\mathcal{D}^{train}$ **do**
6: **for** $t \leftarrow 1, \ldots, T$: a problem solving sequence of student s **do**
7: **if** $(X_t = 1)$ **then** ▷ *correct performance*
8:

$$P_i(L_{t-1}) \leftarrow \frac{P_i(L_{t-1}) * (1 - P_i(\mathbf{S}))}{P_i(L_{t-1}) * (1 - P_i(\mathbf{S})) + (1 - P_i(L_{t-1})) * P_i(\mathbf{G})}$$

9: **else** ▷ *incorrect performance*
10:

$$P_i(L_{t-1}) \leftarrow \frac{P_i(L_{t-1}) * P_i(\mathbf{S})}{P_i(L_{t-1}) * P_i(\mathbf{S}) + (1 - P_i(L_{t-1})) * (1 - P_i(\mathbf{G}))}$$

11: **end if**
12: $P_{si}(L_t) \leftarrow P_i(L_{t-1}) + (1 - P_i(L_{t-1})) * P_i(T)$
13: **end for**
14: **end for**

   /* *Predicting the performance of student s on the tasks required skill i at time $t+1$:* */
15: $\hat{p}^{si}_{t+1} \leftarrow P_{si}(L_t) * (1 - P_i(\mathbf{S})) + (1 - P_{si}(L_t)) * P_i(\mathbf{G})$

16: **end procedure**

where $P(L_{st})$ is determined by using equation 4.2 for a specific student s (this can also be written as $P(X_{t+1}|X_t = x_t, \cdots, X_1 = x_1)$).

Algorithm 1 briefly describes the use of the BKT for predicting student performance. The first step is learning the parameters as presented in line 3. As mentioned before, the LearningParameters procedure can be Expectation Maximization process (as BKT-EM, Chang *et al.* (2006)) or a grid search/brute-force (as BKT-BF, Baker *et al.* (2008b)).

After the parameters are found, the second step is tracing the student's knowledge given their respective data. For example, for each skill, the BKT uses T past performances of a student to compute the "learning probability" of that student (using equation 4.2) as in lines 5-14. Then, this learning parameter is used together with the slip and guess to predict the student's next performance in applying that skill, as in line 15. Please note that the BKT can only deal with only a single skill associated

with the task (for the tasks which have multiple skills, we have to preprocess them as discussed in Section 3.5). Also, the prediction is calculated for each skill i of student s, thus, all tasks (of that student) which require skill i will have the same prediction score.

As we have seen in Algorithm 1, four parameters of the BKT are learned for each skill regardless of the individual students in that skill. This means that the BKT does not take into account "individualization". For example, it does not support students to have different prior knowledge $P(L_0)$ or different learn rates $P(T)$.

To tackle limitations of the BKT, the BKT-PPS (Prior Per Student) (Pardos & Heffernan, 2010), which focuses on individualizing the prior knowledge parameter, was proposed. The difference between the BKT-PPS and the standard BKT is the ability to represent a different prior knowledge parameter for each student, therefore, the $P(L_0)$ in the LearningParameters procedure at line 3 of Algorithm 1 is changed to $P(L_0[s])$ for an individual student s. The $P(L_0[s])$ is then used for the knowledge tracing phase of that student. An illustration of the BKT-PPS is presented in Figure 4.2.

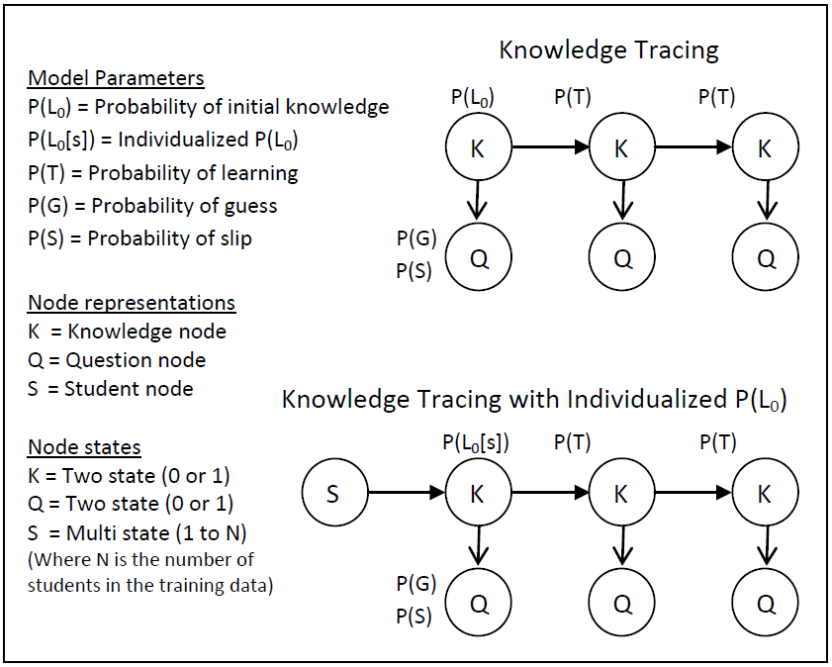


Figure 4.2: Bayesian Knowledge Tracing and Prior Per Student (picture source: Pardos & Heffernan (2010))

Specifically, the BKT-PPS has changed the prior parameter for each student by adding a single node and arc to the standard BKT model (see Figure 4.2). This model has two different student knowledge priors: a high prior and a low prior. The students are assigned to one of those priors based on their first response to the problem

(question). If the students incorrectly answer (their first response) to the problem, they are assumed to be in the low prior group, otherwise they are assumed to be in the high prior group. These priors can be learned or can be set as ad-hoc. Pardos *et al.* (2011) empirically show that, in the ad-hoc heuristic, the high prior should be roughly 1-$P(\mathbf{G})$ and the low prior should be equivalent to the probability of slip $P(\mathbf{S})$.

Another variant of the BKT is BKT-CGS (Contextual Guess and Slip) (Baker *et al.*, 2010). The BKT-CGS contextually estimated whether each individual student response is a guess or a slip, rather than using fixed guess and slip probabilities estimated across all situations. In this method, the guess and slip probabilities are not estimated for each skill, instead, they are computed each time a student attempts to answer a new problem/step, e.g., longer responses and help requests are less likely to be slips.

4.3 Performance Factors Analysis (PFA)

The Performance Factors Analysis (PFA) is another student modeling method which was presented in Pavlik *et al.* (2009). This model is an improvement from previous work called Learning Factor Analysis (LFA) in Cen *et al.* (2006).

First, let us describe the basic form of the LFA. Its model is expressed as the following equation

$$f(s,i) = \alpha_s + \sum_{i \in I} (\beta_i + \gamma_i \cdot A_{s,i}) \tag{4.6}$$

where $f(s,i)$ represents the accumulated learning for student s (ability captured by α_s parameter) using one or more skills $i \in I$ (knowledge components). The easiness of the skill i is captured by the β_i parameters, and the benefit of frequency of prior practice for each skill i is a function $A_{s,i}$ of prior observations for student s with skill i (captured by the addition of the γ_i for each observation) (Pavlik *et al.*, 2009).

Finally, a logistic function is used to convert $f(s,i)$ to the predictions of observed probability, as the following

$$\hat{p}_{si} = \frac{1}{1 + e^{-f(s,i)}} \tag{4.7}$$

The LFA is an elaboration of the Rasch model from Item Response Theory (Rogers *et al.*, 1991), where γ in equation 4.6 is set to 1 and only a single β value is used.

Similar to the LFA, the Performance Factors Analysis (PFA) (Pavlik *et al.*, 2009) also uses a logistic regression model for predicting student performance. The PFA model is presented in the following equation

$$f(s,i) = \sum_{i \in I} (\beta_i + \gamma_i \cdot A_{s,i} + \rho_i \cdot F_{s,i}) \tag{4.8}$$

As in this equation, for each skill i, the PFA predicts student correctness based on the student's number of prior successes $A_{s,i}$ on that skill (which is weighted by a

parameter γ_i) and the student's number of prior failures $F_{s,i}$ on that skill (which is weighted by a parameter ρ_i). The parameter β_i that represents for overall difficulty is also fit for each skill i. Finally, the logistic function in equation 4.7 is still applied to convert to the probability predictions. Since the PFA is an improvement of the LFA by the same group of authors, thus, we only concern the PFA in the remaining discussions.

4.4 Selection of the State-of-the-art Methods

Over the years, many other variants of the BKT and PFA have been proposed. For example, some of the variants of the BKT model can be found in (Baker *et al.*, 2011, 2008a,b, 2010; Beck & Chang, 2007; Chang *et al.*, 2006; Gowda *et al.*, 2011b; Nooraei *et al.*, 2011; Pardos & Heffernan, 2011, 2010; Pavlik *et al.*, 2009; Rai *et al.*, 2009; Ritter *et al.*, 2009; Wang *et al.*, 2011; Wijaya & Prasetyo, 2010) and even more (Conati, 2010; Desmarais & Baker, 2011; Romero *et al.*, 2010); and the other variants of the PFA can be found in (Chi *et al.*, 2011; Gong & Beck, 2011; Gong *et al.*, 2010a; Pavlik *et al.*, 2011).

As we have seen, due to the large number of papers which were varied from the state-of-the-art Bayesian Knowledge Tracing, it is not possible to compare with all of them. Moreover, so far it has been unclear which modeling approach is "the best" in predicting student performance since different comparisons in different cases (e.g., different data sets, different splits, different hyper parameters, etc) have produced different results, and even contradictory findings. For example, Pavlik *et al.* (2009) found that PFA predicts the student performance better than both the BKT-EM and the BKT-BF, and that the BKT-BF performs comparably to or better than the BKT-EM; while Gong *et al.* (2010a) found that the BKT-EM performed equally to the PFA and better than the BKT-BF. In Baker *et al.* (2008b), the authors found that the BKT-EM performed worse than the BKT-BF.

In the followings, we briefly summarize the comparisons presented in the literature, where we denote ">" for "better performance" (please note that these comparisons were conducted on different cases, e.g., data sets, splits, hyper parameters, etc).

- BKT-BF > BKT-EM (Baker *et al.*, 2008b):

  The experiments were conducted using data from an intelligent tutor which covers a wide span of mathematical topics. The data set has 232 students, 581,785 transactions on 171,987 problem steps and 253 skills.

- PFA > {BKT-EM, BKT-BF} (Pavlik *et al.*, 2009):

  Four data sets were used for experiments called Fractions, Algebra, Geometry, and Physics which come from the Carnegie Learning Cognitive Tutor. However, detailed information were not described in the paper.

- BKT-PPS > BKT-EM (Pardos & Heffernan, 2010):

The experiments were conducted on a ASSISTments platform data set having 4,354 students and 42 problem sets with several constraints, e.g., the student have answered all items in the problem set in one day, each problem set has data from at least 100 students, and having at least four items in the problem set of the same skill.

- BKT-EM = PFA > BKT-BF (Gong *et al.*, 2010a,b):

  Data set used for experiments comes from the ASSISTments platform which covers 343 students, 193,259 problem steps and 104 skills.

- BKT-PPS > BKT-BF > BKT-EM > PFA > BKT-CGS (Baker *et al.*, 2011)

  The experiments were conducted on a data set taken from a Cognitive Tutor for Genetics which has 76 students, 11,582 problem steps, 9 skills, and a total of 23,706 transactions.

- BKT-EM > BKT-BF > BKT-PPS > PFA > BKT-CGS (Pardos *et al.*, 2011)

  The experiments were also conducted on the same data set in (Baker *et al.*, 2011). However, the target prediction is on post-test scores rather than within the tutor.

- PFA > BKT-EM > BKT-BF > BKT-PPS > BKT-CGS (Gowda *et al.*, 2011a)

  The experiments were conducted on a data set taken from the ASSISTments platform. There are 178,434 transactions in this data set produced by 5,422 students.

  This work was also conducted by the same group of authors in (Baker *et al.*, 2011; Gong *et al.*, 2010a; Pardos *et al.*, 2011; Pardos & Heffernan, 2010).

  According to the authors *"It is worth noting that while BKT-PPS was the best model in the Pardos & Heffernan (2010), it is the worst model among the BKT models in this work. In general, the relative performance of different student models has been quite unstable between studies. This finding across studies suggests that there is currently no best model; relative model performance appears to be dependent on the data set"* (Gowda *et al.*, 2011a).

Therefore, for comparisons in this thesis, we mainly consider two most popular methods in the family of the state-of-the-art Bayesian Knowledge Tracing, and they also have publicly available codes. Those are BKT-BF[1] (Baker *et al.*, 2008b) and BKT-EM[2] (Chang *et al.*, 2006).

[1] http://users.wpi.edu/∼rsbaker/edmtools.html
[2] http://www.cs.cmu.edu/∼listen/BNT-SM/

4.5 Factorization Techniques

As presented at the beginning, predicting student performance can be considered as rating prediction task in recommender systems, where the *student*, *task*, and *performance* would become the *user*, *item*, and *rating*, respectively. Thus, after this mapping, one can use any method in recommender systems for the predicting student performance problem, e.g., using k-nearest neighbors collaborative filtering (Resnick *et al.*, 1994) or the state-of-the-art matrix factorization (Bell & Koren, 2007; Koren *et al.*, 2009; Paterek, 2007). As already mentioned, this finding is also one of the contributions of this thesis.

We will carefully describe how to apply the state-of-the-art factorization techniques for predicting student performance in Chapters 5 and show that these techniques work quite well on student performance data. They outperform the other methods such as global average, user average, item average, user-item-baseline (Koren, 2010), regularized logistic regressions (Komarek & Moore, 2005), as well as the state-of-the-art Bayesian Knowledge Tracing models (Baker *et al.*, 2008b; Chang *et al.*, 2006; Corbett & Anderson, 1995).

By now, there is only one published work proposed by Töscher & Jahrer (2010) who have applied factorization models for predicting student performance. However, this work is also a parallel work with the proposed methods in this thesis. The authors aimed at the KDD Challenge 2010, thus, in that work they proposed extending the factorization methods to several complex models, making feature engineering on 105 features, and finally, combining all of them with an ensemble model, which obviously have better performance than our proposed methods.

As discussed at the beginning, our work aims at contributing a new approach for the Student Modeling communities, especially to the educational data miners who are priorly unaware of these techniques, rather than producing a system for the KDD Challenge 2010. However, more sophisticated exploiting the proposed approach can archive further improvements, as shown in Töscher & Jahrer (2010).

Chapter 5

Student Modeling with Latent Factor Models

Contents

5.1 Introduction 39
5.2 Modeling Student/Task Latent Factors 41
5.2.1 Training Phase 42
5.2.2 Prediction Phase 44
5.2.3 An Example 44
5.3 Modeling Student/Task Effects (Biases) 45
5.4 Modeling Student/Task Correlations 47
5.4.1 User-based Collaborative Filtering (User-kNN) 48
5.4.2 Item-based Collaborative Filtering (Item-kNN) 48
5.5 Experiments 49
5.5.1 Software implementations 49
5.5.2 Experimental Setting 50
5.5.3 Experimental Results 52
5.6 Discussion 56

5.1 Introduction

Usually, information acquisition is a challenging problem in many real-world applications since the collection of attributes and meta data about the users, tasks and/or other resources is often very expensive or even not possible at all.

In predicting student performance, the easily obtainable information is the results of the tasks on which the students previously worked. These information could be captured by the {*student, task, score*} triple. Interestingly, this is exactly the kind of information that the current recommender systems rely on. In the recommender system terminology the above mentioned tuple would be rephrased as {*user, item, rating*}.

This perspective allows two distinct kinds of treatment. First, the so-called "implicit feedback" where only the information that a specific student (user) interacted with a given task (item) via reading, clicking, viewing, etc. are recorded. These interactions can be represented by a binary or even unary matrix. Second, the "explicit feedback" where a numerical score (performance/preference) of interaction between student and task can be obtained. These, too, can be represented as an interaction matrix on the students and the tasks. In both cases, however, the resulting matrix will be very sparse, as most of the students may solve only a (small) subset of the tasks/exercises which are available in the tutoring system.

As already mentioned, in the recommender system context, predicting student performance can be considered as a rating prediction problem since *student, task*, and *performance* information can be treated as *user, item,* and *rating*, respectively, which are the main objects recommender systems learn from, nowadays.

One of the state-of-the-art methods in recommender systems, which can be used for both item and rating prediction, and thus for predicting student performance, is the matrix factorization (Bell & Koren, 2007; Koren *et al.*, 2009; Rendle & Schmidt-Thieme, 2008). It has been shown that even a few ratings are often more valuable than meta data (Pilászy & Tikk, 2009), and in case of sparse data with no additional meta data, the use of matrix factorization techniques has been shown to be very effective.

Indeed, factorization techniques outperform other state-of-the-art collaborative filtering techniques. They belong to the family of latent factor models which aim at mapping users and items to a common latent space by representing them as vectors in that space. The dimensions of this space are called the factors. Here we should mention that we usually do not now the exact "meaning" of these factors and we are just interested in the correlation between the vectors in that space. For example, imagine a well-known simplified movie rating example where the users and the movies are mapped to a two-dimensional latent space (Koren *et al.*, 2009). Here, each user or each movie is represented by two factors. These factors can represent genre, seriousness, amount of action, quality of actors or any other concept. Even though we do not know what exactly the given two factors (dimensions of the latent space) represent, the "closeness" of particular user and movie vectors in that space expresses the preference of a given user on the given movie. Thus, we can expect that the aforementioned *slip* and *guess* factors are implicitly encoded in the latent factors of the factorization models and we may not need to explicitly take care of them as in other methods, e.g., in the Bayesian Knowledge Tracing models (Baker *et al.*, 2008b; Chang *et al.*, 2006; Corbett & Anderson, 1995; Pardos & Heffernan, 2010).

Moreover, with latent factor models, the other latent characteristics of students

and tasks could also be implicitly encoded in the models such as how good/clever/bad the student is, how hard/easy the task is, etc. As also mentioned in Pilászy & Tikk (2009), these techniques can work well in case of using just two features such as user ID (student) and item ID (task). Thus, memory consumption and the human effort in data pre-processing can be reduced significantly while the prediction quality is reasonable.

Please note that the techniques presented in this chapter compute the prediction as a linear combination of the latent factors. Although it may happen that in some cases a linear combination of the factors may be insufficient. In such cases, we can use non-linear extensions of the factorization techniques, e.g., in Lawrence & Urtasun (2009). However, as showed in Takács *et al.* (2009), using the standard factorization models is enough to reach good prediction accuracy in an efficient and scalable way. The results of this approach in our later experiments as well as the experimental results in other domains (Koren *et al.*, 2009) constitute empirical evidences that assuming a linear interaction between the factors is a reasonable approach. The connection between the factorization techniques and other methods, such as Neural Networks, can be seen, e.g., in Takács *et al.* (2009).

We will describe the standard factorization techniques to present the idea behind these approaches. Discussion of other advanced factorization techniques is out of the scope of this chapter.

5.2 Modeling Student/Task Latent Factors

As mentioned at the beginning, the latent factor models, especially the state-of-the-art matrix factorization, would be a good choice for student modeling, specifically for predicting student performance. We thoroughly describe this technique in the following.

The idea of matrix factorization (MF) is to approximate a matrix $\mathbf{X} \in \mathbb{R}^{|S|\times|I|}$ by the product of two smaller matrices $\mathbf{W}$ and $\mathbf{H}$, i.e.

$$\mathbf{X} \approx \mathbf{W}\mathbf{H}^T$$

where $\mathbf{W} \in \mathbb{R}^{|S|\times K}$ is a matrix where each row s is a vector containing K latent factors describing the student s, and $\mathbf{H} \in \mathbb{R}^{|I|\times K}$ is a matrix where each row i is a vector containing K latent factors describing the task i.

Let w_{sk} and h_{ik} be the elements and w_s and h_i the vectors of $\mathbf{W}$ and $\mathbf{H}$, respectively, then the performance p given by a student s to a task i is predicted by:

$$\hat{p}_{si} = \sum_{k=1}^{K} w_{sk} h_{ik} = w_s h_i^T \tag{5.1}$$

The main issue of this technique is how to find optimal values for the parameters $\mathbf{W}$ and $\mathbf{H}$ given a criterion such as Root Mean Squared Error (RMSE), which is determined by

$$\text{RMSE} = \sqrt{\frac{\sum_{(s,i,p)\in\mathcal{D}^{test}}(p_{si} - \hat{p}_{si})^2}{|\mathcal{D}^{test}|}} \tag{5.2}$$

5.2.1 Training Phase

Using matrix factorization, training the model is to find the optimal parameters $\mathbf{W}$ and $\mathbf{H}$. One approach is that we first initialize these two matrices with some random values, e.g., from the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean = 0 and standard deviation σ^2 = 0.01, and compute the error (objective) function, for example

$$\mathcal{O}^{MF} = \sum_{(s,i,p)\in\mathcal{D}^{train}} e_{si}^2 \tag{5.3}$$

where

$$e_{si}^2 = (p_{si} - \hat{p}_{si})^2 = (p_{si} - \sum_{k=1}^{K} w_{sk}h_{ik})^2 \tag{5.4}$$

then try to minimize this error function by updating the values of $\mathbf{W}$ and $\mathbf{H}$ iteratively, e.g., using gradient descent (Bell & Koren, 2007; Takács *et al.*, 2007).

To minimize the error function in equation (5.3), we need to know for each data point in which direction to update the value of w_{sk} and h_{ik}. Thus, we compute the gradient of the function (5.4):

$$\frac{\partial}{\partial w_{sk}} e_{si}^2 = -2e_{si}h_{ik} = -2(p_{si} - \hat{p}_{si})h_{ik} \tag{5.5}$$

$$\frac{\partial}{\partial h_{ik}} e_{si}^2 = -2e_{si}w_{sk} = -2(p_{si} - \hat{p}_{si})w_{sk} \tag{5.6}$$

After having the gradients, we update the values of w_{sk} and h_{ik} in the direction opposite to the gradient:

$$w'_{sk} = w_{sk} - \beta\frac{\partial}{\partial w_{sk}} e_{si}^2 = w_{sk} + 2\beta e_{si}h_{ik} = w_{sk} + 2\beta(p_{si} - \hat{p}_{si})h_{ik} \tag{5.7}$$

$$h'_{ik} = h_{ik} - \beta\frac{\partial}{\partial h_{ik}} e_{si}^2 = h_{ik} + 2\beta e_{si}w_{sk} = h_{ik} + 2\beta(p_{si} - \hat{p}_{si})w_{sk} \tag{5.8}$$

where β is the learning rate.

We iteratively update the values of $\mathbf{W}$ and $\mathbf{H}$ until the error converges to its minimum ($\mathcal{O}^{\text{MF}}_{Iter_{(n-1)}} - \mathcal{O}^{\text{MF}}_{Iter_n} < \epsilon$) or reaching a predefined number of iterations.

Regularization term

To prevent over-fitting, we modify the error function (5.4) by adding a term which controls the magnitudes of the factor vectors such that **W** and **H** would give a good approximation of **X** without having to contain large numbers. The error function now becomes:

$$\mathcal{O}^{MF} = \sum_{(s,i,p)\in\mathcal{D}^{train}} (p_{si} - \sum_{k=1}^{K} w_{sk}h_{ik})^2 + \lambda \cdot (||\mathbf{W}||_F^2 + ||\mathbf{H}||_F^2) \tag{5.9}$$

where $||\cdot||_F$ is a Frobenius norm[1] and λ is a regularization term (regularization weight).

With this new error function, the values of w_{sk} and h_{ik} are updated by

$$w'_{sk} = w_{sk} + \beta(2e_{si}h_{ik} - \lambda w_{sk}) = w_{sk} + \beta(2(p_{si} - \hat{p}_{si})h_{ik} - \lambda w_{sk}) \tag{5.10}$$

$$h'_{ik} = h_{ik} + \beta(2e_{si}w_{sk} - \lambda h_{ik}) = h_{ik} + \beta(2(p_{si} - \hat{p}_{si})w_{sk} - \lambda h_{ik}) \tag{5.11}$$

Recall that there are two issues that should be taken into account when predicting student performance, namely the "guess" and "slip" factors expressing the probabilities that the students will guess correctly or make a mistake. Matrix factorization would be an appropriate approach for handling these issues because the mentioned "slip" and "guess" factors could be implicitly encoded in the latent factors of **W** and **H**.

Algorithm 2 describes details of training a matrix factorization model using stochastic gradient descent (we use stochastic gradient descent for all algorithms in our work since it has been shown that the computing cost of stochastic gradient descent has a huge advantage for large-scale problems (Bottou, 2004)).

First, the parameters **W** and **H** are initialized randomly from the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean is 0 and standard deviation $\sigma^2 = 0.01$, as in lines 2-3. While the stopping condition is not met, e.g., reaching the maximum number of predefined iterations or converging ($\mathcal{O}^{\text{MF}}_{Iteration_{(n-1)}} - \mathcal{O}^{\text{MF}}_{Iteration_n} < \epsilon$), the latent factors are updated iteratively. For example, in each iteration, we randomly select an instance in the training set (s, i, p), then compute the prediction for this student and task, as in lines 5-9. We then estimate the error in this iteration and update the values of **W** and **H** as in lines 11-14.

[1] Frobenius norm of the matrix **W** is determined by (see more at http://en.wikipedia.org/wiki/Matrix_norm#Frobenius_norm):

$$||\mathbf{W}||_F = \sqrt{\sum_{s=1}^{|S|}\sum_{k=1}^{K}|w_{sk}|^2}$$

Algorithm 2 Learn a matrix factorization for factorizing student and task using stochastic gradient descent with K latent factors, β learning rate, λ regularization term, and stopping criterion

procedure STUDENT-TASK-MATRIXFACTORIZATION($\mathcal{D}^{train}$, K, β, λ, stopping condition)
  Let $s \in S$ be a student, $i \in I$ a task, $p \in P$ a performance score
  Let $W[|S|][K]$ and $H[|I|][K]$ be latent factors of students and tasks
  $W \leftarrow \mathcal{N}(0, \sigma^2)$
  $H \leftarrow \mathcal{N}(0, \sigma^2)$
  while (Stopping criterion is NOT met) **do**
    Draw randomly (s, i, p) from $\mathcal{D}^{train}$
    $\hat{p} \leftarrow 0$
    for $k \leftarrow 1, \ldots, K$ **do**
      $\hat{p} \leftarrow \hat{p} + W[s][k] * H[i][k]$
    end for
    $e_{si} = p - \hat{p}$
    for $k \leftarrow 1, \ldots, K$ **do**
      $W[s][k] \leftarrow W[s][k] + \beta * (e_{si} * H[i][k] - \lambda * W[s][k])$
      $H[i][k] \leftarrow H[i][k] + \beta * (e_{si} * W[s][k] - \lambda * H[i][k])$
    end for
  end while
  return $\{W, H\}$
end procedure

5.2.2 Prediction Phase

After the training phase, we have the two optimal latent factors **W** and **H**, the remaining task is straightforward. The performance of a student s in a given task i is predicted easily by equation 5.1.

Please note that, for the new students or the new tasks, those are in the test set but not in the train set, we can simply return the global average (average performance of all students in the training set). We will discuss more about this problem in Section 5.5.2.

5.2.3 An Example

Figure 5.1 shows an example of how we can factorize the students and tasks. Suppose that we have six students and five exercises (tasks) which are presented in a matrix **X**. Each task is to compute the values of y, e.g. $y = -2x$, given a specific value of x. These students have performed some tasks which are measured by correct (1) or incorrect (0) performance. Our problem is to predict the other tasks that they have not done (the empty values in **X**).

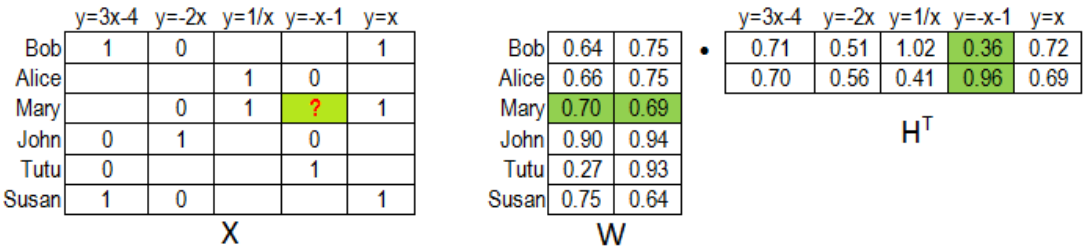


Figure 5.1: An example of factorizing on students and tasks

Please note that the presented techniques do not depend on specific tasks. Importance is that the matrix represents a relation between two types of objects, which are in our case students and math tasks but any other types of objects can be also used, e.g., students and courses, students and learning resources (books or multimedia, etc.) as well as tasks in other disciplines.

After the training phase with $K = 2$ latent factors, we get the optimized parameters $\mathbf{W}$ and $\mathbf{H}$ as in this figure. Now, suppose that we would like to predict Mary's performance for the task $y = -x - 1$ (the cell with question mark). We can easily compute the prediction using equation (5.1)

$$\hat{p}_{si} = \sum_{k=1}^{K} w_{sk} h_{ik} = 0.7 * 0.36 + 0.69 * 0.96 = 0.91$$

From this prediction result, we can see that Mary may correctly answer her task with high confidence (0.91). In a similar way, we can predict the performance of other students in tasks which they have not done yet.

Please note that it may happen the cases that $\hat{p}_{si} < 0$ or $\hat{p}_{si} > 1$. In those cases, we simply bound the prediction results in 0 or 1.

5.3 Modeling Student/Task Effects (Biases)

In previous section, we have presented using the standard matrix factorization to encode the student/task latent factors. In this section, we introduce how to use the biased matrix factorization (BMF) to deal with the problem of "user effect" ("user bias") and "item effect" ("item bias") (Koren *et al.*, 2009).

On the educational setting, the user and item biases are, respectively, the *student* and *task* biases/effects. The student effect (student bias) models how good/clever/bad a student is (i.e., how likely is the student to perform a task correctly), and the task effect (task bias) models how difficult/easy the task is (i.e., how likely is the task to be performed correctly).

Using these biases, the performance of student s for a given task i is now determined by

$$\hat{p}_{si} = \mu + b_s + b_i + \sum_{k=1}^{K} w_{sk} h_{ik} \tag{5.12}$$

where μ is a global average (average performance of all students and tasks in the training set $\mathcal{D}^{train}$)

$$\mu = \frac{\sum_{(s,i,p)\in\mathcal{D}^{train}} p}{|\mathcal{D}^{train}|} \tag{5.13}$$

b_s is the student bias (average performance of student s deviated from the global average)

$$b_s = \frac{\sum_{(s',i,p)\in\mathcal{D}^{train}|s'=s} (p-\mu)}{|\{(s',i,p)\in\mathcal{D}^{train}|s'=s\}|} \tag{5.14}$$

and b_i is the task bias (average performance on task i deviated from the global average)

$$b_i = \frac{\sum_{(s,i',p)\in\mathcal{D}^{train}|i'=i} (p-\mu)}{|\{(s,i',p)\in\mathcal{D}^{train}\}|i'=i|} \tag{5.15}$$

Moreover, the error function is now changed by adding these two biases to the regularization:

$$\mathcal{O}^{BMF} = \sum_{(s,i,p)\in\mathcal{D}^{train}} (p_{si} - \mu - b_s - b_i - \sum_{k=1}^{K} w_{sk} h_{ik})^2 + \lambda(||\mathbf{W}||_F^2 + ||\mathbf{H}||_F^2 + b_s^2 + b_i^2) \tag{5.16}$$

Algorithm 3 describes more details about using biased matrix factorization for predicting student performance. First, we compute the global average, student bias and task bias as in lines 2-8. The parameters $\mathbf{W}$ and $\mathbf{H}$ are again initialized randomly from the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean is 0 and standard deviation $\sigma^2 = 0.01$. Then, we update the value of μ, b_s, b_i, $\mathbf{W}$ and $\mathbf{H}$ at each iteration as in lines 11-21. After getting these parameters, we can easily compute the prediction in the test set for any existing student and task with the same formula in line 13.

Similar to the matrix factorization, after the training phase, we achieve the optimal two latent factors $\mathbf{W}$ and $\mathbf{H}$ and the biased terms. The performance of a student s for a given task i is easily predicted by using equation (5.12).

Algorithm 3 Learn a biased matrix factorization for factorizing student and task using stochastic gradient descent with K latent factors, β learning rate, λ regularization weight, and stopping condition

procedure STUDENT-TASK-BIASEDMATRIXFACTORIZATION($\mathcal{D}^{train}$, K, β, λ, stopping condition)
  Let $s \in S$ be a student, $i \in I$ a task, $p \in P$ a performance score
  Let $W[|S|][K]$ and $H[|I|][K]$ be latent factors of students and tasks
  Let $b_s[|S|]$ and $b_i[|I|]$ be student-bias and task-bias
  $\mu \leftarrow \frac{\sum_{p \in \mathcal{D}^{train}} p}{|\mathcal{D}^{train}|}$
  for each student s **do**
    $b_s[s] \leftarrow \frac{\sum_i (p_{si} - \mu)}{|\mathcal{D}_s^{train}|}$
  end for
  for each task i **do**
    $b_i[i] \leftarrow \frac{\sum_u (p_{si} - \mu)}{|\mathcal{D}_i^{train}|}$
  end for
  $W \leftarrow \mathcal{N}(0, \sigma^2)$
  $H \leftarrow \mathcal{N}(0, \sigma^2)$
  while (Stopping criterion is NOT met) **do**
    Draw randomly (s, i, p_{si}) from $\mathcal{D}^{train}$
    $\hat{p}_{si} \leftarrow \mu + b_s[s] + b_i[i] + \sum_k^K (W[s][k] * H[i][k])$
    $e_{si} = p_{si} - \hat{p}_{si}$
    $\mu \leftarrow \mu + \beta * e_{si}$
    $b_s[s] \leftarrow b_s[s] + \beta * (e_{si} - \lambda * b_s[s])$
    $b_i[i] \leftarrow b_i[i] + \beta * (e_{si} - \lambda * b_i[i])$
    for $k \leftarrow 1, \ldots, K$ **do**
      $W[s][k] \leftarrow W[s][k] + \beta * (e_{si} * H[i][k] - \lambda * W[s][k])$
      $H[i][k] \leftarrow H[i][k] + \beta * (e_{si} * W[s][k] - \lambda * H[i][k])$
    end for
  end while
  return $\{W, H, b_s, b_i, \mu\}$
end procedure

5.4 Modeling Student/Task Correlations

In recommender system context, we usually assume that "similar users" may like "similar items" and vice versa. Similarly, in our educational domain, we also assume that "similar students" may have similar performances on "similar tasks". Thus, user-based/ item-based collaborative filtering would be a choice for taking into account correlations between the students and the tasks in predicting student performance. We will briefly describe how to use the (k-nearest neighbors) collaborative filtering in the following sections (see details in, e.g., Resnick *et al.* (1994); Sarwar *et al.* (2001); Su & Khoshgoftaar

(2009)).

5.4.1 User-based Collaborative Filtering (User-kNN)

In this approach, the predicted performance $\hat{p}_{si}$ of student s on task i is based on the performances of its nearest neighbors (students) on that task. The prediction using weighted sum is determined by

$$\hat{p}_{si} = \frac{\sum_{s' \in K_s} sim(s, s') p_{s'i}}{\sum_{s' \in K_s} |sim(s, s')|} \tag{5.17}$$

where K_s is the set of K nearest neighbors of student s, and $sim(s, s')$ is the similarity between student s and student s' which can be computed by using the *Cosine similarity* or *Pearson similarity*

$$sim_{pearson}(s, s') = \frac{\sum_{i \in I_{ss'}} (p_{si} - \bar{p}_s)(p_{s'i} - \bar{p}_{s'})}{\sqrt{\sum_{i \in I_{ss'}} (p_{si} - \bar{p}_s)^2 \sum_{i \in I_{ss'}} (p_{s'i} - \bar{p}_{s'})^2}} \tag{5.18}$$

$$sim_{cosine}(s, s') = \frac{\sum_{i \in I_{ss'}} p_{si} p_{s'i}}{\sqrt{\sum_{i \in I_{ss'}} p_{si}^2 \sum_{i \in I_{ss'}} p_{s'i}^2}} \tag{5.19}$$

where $I_{ss'}$ is a set of tasks performed by both student s and student s'; and $\bar{p}_s$ and $\bar{p}_{s'}$ are the mean (average) performance over all the tasks of student s and s', respectively.

Another prediction approach, instead of using the weighted sum, one could also use the prediction using deviations from the user (student) mean. Using deviation, the performance of student s on task i is now determined by

$$\hat{p}_{si} = \bar{p}_s + \frac{\sum_{s' \in K_s} sim(s, s')(p_{s'i} - \bar{p}_{s'})}{\sum_{s' \in K_s} |sim(s, s')|} \tag{5.20}$$

We will call this approach "Student-kNN".

5.4.2 Item-based Collaborative Filtering (Item-kNN)

Similar to the user-based approach, however, instead of using user similarity, the item-based approach uses item similarity to make prediction. The prediction performance $\hat{p}_{si}$ of a student s on a task i using weighted sum is determined by

$$\hat{p}_{si} = \frac{\sum_{i' \in K_i} sim(i, i') p_{si'}}{\sum_{i' \in K_i} |sim(i, i')|} \tag{5.21}$$

where K_i is the set of K nearest neighbors of task i, which are performed by student s, and $sim(i, i')$ is the similarity between task i and task i' which can be computed by using the *Cosine* similarity or *Pearson* similarity

$$sim_{pearson}(i, i') = \frac{\sum_{s \in S_{ii'}} (p_{si} - \bar{p}_i)(p_{si'} - \bar{p}_{i'})}{\sqrt{\sum_{s \in S_{ii'}} (p_{si} - \bar{p}_i)^2 \sum_{s \in S_{ii'}} (p_{si'} - \bar{p}_{i'})^2}} \tag{5.22}$$

$$sim_{cosine}(i, i') = \frac{\sum_{s \in S_{ii'}} p_{si} p_{si'}}{\sqrt{\sum_{s \in S_{ii'}} p_{si}^2 \sum_{s \in S_{ii'}} p_{si'}^2}} \tag{5.23}$$

where $S_{ii'}$ is a set of students who perform both task i and task i'; and $\bar{p}_i$ and $\bar{p}_{i'}$ are the mean (average) performance over all students of task i and i', respectively.

The prediction performance using deviations from the mean is determined by

$$\hat{p}_{si} = \bar{p}_i + \frac{\sum_{i' \in K_i} sim(i, i')(p_{si'} - \bar{p}_{i'})}{\sum_{i' \in K_i} |sim(i, i')|} \tag{5.24}$$

where $\bar{p}_i$ and $\bar{p}_{i'}$ are the mean (average) performance on task i and i', respectively. We will call this approach "Task-kNN".

It has been shown that prediction using deviations from the user/item mean is more accurate than prediction using the weighted sum (Adomavicius & Tuzhilin, 2005; Herlocker *et al.*, 1999). Thus, in this chapter we use the deviations from the user/item mean for the predictions in all later experiments.

5.5 Experiments

In this section, we first describe software implementations and experimental setting. We then present experimental results. Finally, some discussions are presented.

5.5.1 Software implementations

We have implemented the softwares for all the proposed methods including the latent factor models in this chapter, the multi-relational factorization models in Chapter 6, the personalized forecasting methods in Chapter 7, and the tensor factorization models in Chapter 8. Moreover, we have also implemented the other baselines, such as *global average*, *student average* (this is *user average* in recommender systems (Vozalis & Margaritis, 2003)), *biased student-task* which is originally from the *baseline predictor* (user-item baseline) in Koren & Bell (2011), and the Correct First Attempt Rates (CFAR) (Yu *et al.*, 2010). These softwares are written in Java codes and parts of them are publicly available at http://www.ismll.uni-hildesheim.de/personen/nguyen_en.html. However,

for a faster and full-fledged version, we point the interested readers to MyMediaLite (Gantner *et al.*, 2011), which is an open source recommender systems library written in C-Sharp and is available at http://www.ismll.uni-hildesheim.de/mymedialite/index.html.

For the Bayesian Knowledge Tracing BKT-BF (Baker *et al.*, 2008b; Corbett & Anderson, 1995), we have used the published software written in Java codes by Baker *et al.* (2008b), which is available at http://users.wpi.edu/∼rsbaker/edmtools.html.

For the Bayesian Knowledge Tracing BKT-EM (Chang *et al.*, 2006; Corbett & Anderson, 1995), we have used the published software written in Matlab by Chang *et al.* (2006), which is publicly available at http://www.cs.cmu.edu/∼listen/BNT-SM/.

For logistic regression, we have used the Truncated Regularized Iteratively Reweighted Least Squares (LR-TRIRLS) by Komarek & Moore (2005), which is publicly available at http://komarix.org/ac/lr/lrtrirls.

5.5.2 Experimental Setting

Data sets

We use the data sets as described in Chapter 3. The information of *user* (student), *item* (task), and *rating* (performance) have already been described in Table 3.3.

Baselines

The proposed methods are compared with the simple baseline *global average*, *student average*, *biased student-task* (*baseline predictor* in Koren & Bell (2011)), and the CFAR (Yu *et al.*, 2010).

Precisely, for the *global average*, we simply compute the average performances on the training set (in equation 5.13), then, using this number as the predicted values for all instances in the test set.

The *student average* is similar to the global average but averaging for each individual student, as in the following equation:

$$\hat{p}_s = \frac{\sum_{(s',i,p)\in\mathcal{D}^{train}|s'=s} p}{|\{(s',i,p) \in \mathcal{D}^{train}|s' = s\}|} \tag{5.25}$$

For the *biased-student-task*, we predict the performance of each student s and task i using the following equation

$$\hat{p}_{si} = \mu + b_s + b_i \tag{5.26}$$

where μ is still the global average; b_s is the student bias; and b_i is the task bias, as described in equations 5.13, 5.14, 5.15, respectively.

However, as presented in Koren & Bell (2011), by adding regularization terms to these biases to prevent over-fitting one may improve the prediction results, thus, we have modified these biases to come up with the following equations:

$$b_i = \frac{\sum_{(s,i',p)\in\mathcal{D}^{train}|i'=i}(p-\mu)}{|\{(s,i',p)\in\mathcal{D}^{train}|i'=i\}|+\lambda_{b_i}} \tag{5.27}$$

$$b_s = \frac{\sum_{(s',i,p)\in\mathcal{D}^{train}|s'=s}(p-\mu-b_i)}{|\{(s',i,p)\in\mathcal{D}^{train}|s'=s\}|+\lambda_{b_s}} \tag{5.28}$$

where λ_{b_s} and λ_{b_i} are the regularization terms for student s and task i, respectively.

Moreover, we also compare the proposed models with traditional methods such as regularized logistic regression (Komarek & Moore, 2005). Of course, the data sets need to be transformed for regression, for example, using the method described in Section 3.3.

Furthermore, the proposed methods are also compared with the state-of-the-art methods in student modeling: BKT-BF (Baker *et al.*, 2008b; Corbett & Anderson, 1995) and BKT-EM (Chang *et al.*, 2006; Corbett & Anderson, 1995).

Evaluation schema

Root mean squared error (RMSE) is used for evaluation. We would like to simulate the prediction results of the proposed methods by using a real system from KDD Challenge 2010 to see how far our models can improve compared to the others on the given data sets. Thus, the RMSE reported in this study are obtained from this KDD Challenge 2010 website[1] (it is still opened for submission after the challenge by now).

Hyper parameter setting

For the regularization values of the *biased student-task*, based on preliminary results on these data sets, we found that the results are not significantly different on different regularization values, so we set where $\lambda_{b_s} = 5$ and $\lambda_{b_i} = 10$.

For the latent factor models and the other baselines, the hyper parameter search is also applied (e.g., optimizing the RMSE on the validation set). However, due to the large spaces of the hyper parameters, we have just done a raw search for the proposed methods, e.g.

$$\beta \in (10^{-4}, 10^{-3}, 10^{-2}, 5\cdot10^{-5}, 5\cdot10^{-4}, 5\cdot10^{-3}),$$

$$\lambda \in (15\cdot10^{-4}, 15\cdot10^{-3}, 55\cdot10^{-5}, 55\cdot10^{-4}, 55\cdot10^{-3}), \text{ and } K \in (2^4, \ldots, 2^8)$$

. The number of iterations depend on each data set, e.g. the algorithms stop iterating when converging or over-fitting. Other choices may produce better results, though.

[1] https://pslcdatashop.web.cmu.edu/KDDCup/

For the kNN collaborative filtering, from the preliminary results on these data sets, we found that "the larger the k, the better the result", thus, we set the k to the maximum value by using all student (task) nearest-neighbors in K_s (K_i), for example,

$$\forall s' \in K_s | sim(s, s') > 0$$

$$\forall i' \in K_i | sim(i, i') > 0$$

We also treat the similarities *Cosine* and *Pearson* as the hyper parameters.

Moreover, four parameters of the Bayesian Knowledge Tracing (prior knowledge $P(L_0)$, learn rate $P(T)$, slip $P(S)$, and guess $P(G)$) also need to be determined. For the BKT-BF, it uses an exhaustive search to determine these parameters. First, it starts a coarse search from 0.01 to 0.99 with the increment of 0.01. After these parameters are found, a fine-grained search is again applied (from -0.009 to 0.009 with the increment of 0.001). For the BKT-EM, the parameters were initialized similar to the other literature, e.g., in (Baker *et al.*, 2011; Gong *et al.*, 2010a): $P(L_0) = 0.5$, $P(T) = 0.14$, $P(S) = 0.09$, and $P(G) = 0.14$. Then, these parameters were adjusted during the expectation-maximization process.

Dealing with cold-start problem

One of the challenging problem of collaborative filtering approaches like (biased) matrix factorization is to deal with the "new user" (new student) or "new item" (new task), e.g., those that are in the test set but not in the train set. We treat this problem with a simple strategy: providing the global average score for the new users or new items. Of course, using more sophisticated methods, e.g., in Gantner *et al.* (2010a); Preisach *et al.* (2010), can improve the prediction results. However, in our work, these tasks are not the problems we would like to tackle, so we leave them for future work.

Furthermore, in the educational data mining scenario, the cold-start problem is not as harmful as in the e-commerce environment where the new users and new items appear every day or even hour, and thus, the models need not to be re-trained continuously.

5.5.3 Experimental Results

As already outlined in Section 3.2, one of the central issues in mapping the student performance prediction to a recommender system problem is determining exactly what set of attributes describes an *item* in a recommender system setting, and there are several choices for this mapping. The first obvious choice for an item is the set of attributes that uniquely describes a task performed by a student which was already denoted as *solving-step* (I_7) in Tables 3.2 and 3.3. However, we now investigate on other choices to understand which *item* can give the better results.

Table 5.1 shows the root mean squared error (RMSE) of using different sets of attributes as items, evaluated by using Matrix Factorization. Besides the solving-step (I_7), we have also investigated on *Problem Group* (I_5), *Problem Name* (I_2), *Step Name*

(I_3), and *problem-step* ($I_2 + I_3$) since with them we can reduce the new items and the sparsities. The results of using other configurations as "items" did not show significant improvement, so we do not report on them.

Table 5.1: RMSE of using different sets of attributes as items (tasks)

Item (Task)	**Algebra**	**Bridge**	**Average**
Problem Group (I_5)	0.33282	0.31572	0.32427
Problem Name (I_2)	0.32705	0.30351	0.31528
Step Name (I_3)	0.32122	0.29440	0.30781
Problem-step ($I_2 + I_3$)	0.31338	0.29322	0.30330
Solving-step (I_7)	0.31293	0.29353	**0.30323**

We observed that using the *solving-step* instead of just Problem Group or Problem Name as an item yields significantly better results for matrix factorization. This has to do with the fact that the solving-step is a combination of four features which describe one single try of one student, thus being more meaningful for predicting his/her performance than just the problem group or the name of the problem he was working on. Moreover, using solving-step as an item can make the prediction less ambiguous than the others since our target is to predict a single step (Problem Name $\subseteq$ Problem Group which covers many single steps). However, for this choice, we have to face with the sparsity in the data sets.

Figure 5.2 shows the RMSE scores of the proposed methods compared to the others, which we use solving-step (I_7) as the task. We observed that the latent factor models (matrix factorization and biased matrix factorization) outperform the other methods.

The logistic regression outperforms the other baselines: global average, student average, and biased-student-task. In preliminary works, we found that the results of linear regression and logistic regression are very similar, we just report on logistic regression here. (please note that the experiments with logistic regression were carried out using the mapping of two attributes (Student Average, solving-step Average) as described in Section 3.3).

Since the idea of predicting students performance is to find out whether a student has learned the knowledge required to solve a certain task, which is represented by the knowledge components (KC) or the skills, we also report the results of using the skill as an item (I_6 in Table 3.3). In this case, instead of directly predicting student performance on a particular task, we predict the student performance on the required skills associated with the task.

Figure 5.3 shows the RMSE scores of the proposed methods compared to the others, which we use the skill (I_6) as the task. Clearly, the latent factor models significantly improve the results on the Algebra data set. The reason why the latent factor models do not work well on the Bridge data set is that the skill (KC-rules) was not provided by the

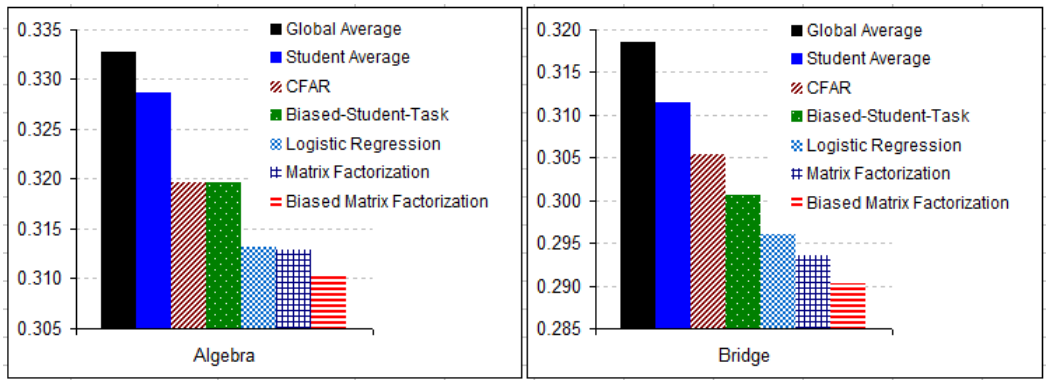


Figure 5.2: RMSE: Latent factor models vs. others (solving-step is used as task)

organizers, instead, we have used the KC-Trace which is less informative (Koedinger *et al.*, 2010), as already described in Chapter 3 (please also note that, in this case, the experiments with logistic regression were carried out using the mapping of two respective attributes (Student Average, Skill Average)).

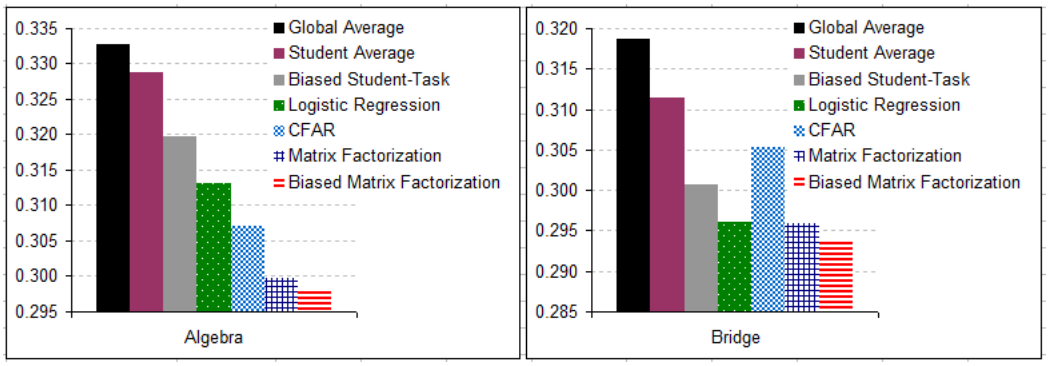


Figure 5.3: RMSE: Latent factor models vs. others (skill is used as task)

Moreover, the use of skill as an item is similar to the state-of-the-art Bayesian Knowledge Tracing models which trace how students apply their gained knowledge / skill on the given tasks. Thus, we now compare the proposed methods with the BKT-BF (Baker *et al.*, 2008b) and the BKT-EM (Chang *et al.*, 2006) on these data to understand how far our models can improve. However, the BKT-EM runs rather slow on these large data sets, even it is intractable on the Bridge, so, we did not report the BKT-EM on the Bridge data set. The comparative results are presented in Table 5.2. Here, we can see that the proposed collaborative filtering methods (Student-kNN, Task-kNN and latent factor models) are also improved to the state-of-the-art BKT models.

For the latent factor models, using the skill as an item achieving better results

Table 5.2: RMSE: Collaborative Filtering Approaches vs. Bayesian Knowledge Tracing

Method	**Algebra**	**Bridge**	**Average**	**ASSISTments**
Knowledge Tracing - EM	0.31098	N/A	N/A	0.48860
Knowledge Tracing - BF	0.31308	0.30849	0.31078	0.49353
Student-kNN-Cosine	0.30699	0.30168	0.30434	0.48431
Task-kNN-Cosine	0.30221	0.29979	0.30100	0.47317
Matrix Factorization	0.29898	0.29446	0.29672	0.46041
Biased Matrix Factorization	0.29819	0.29385	**0.29602**	**0.45822**

than using the solving-step as an item, on the Algebra data set. The reason for this improvement could be that, in this setting, there are less items/tasks (2,979), thus, on average, each user (student) has more ratings (8,918,054/2,979 $\approx$ 2,994 ratings/user), which means that the algorithms have more data to learn from.

Moreover, by employing the student effect and task effect to the latent factor model (biased matrix factorization) we can achieve further improvements. These results validate for the issues that we have mentioned before, that the latent factor models may implicitly encode the student/task latent factors (e.g. "slip", "guess") into their models.

For referencing, we report the hyper parameters found and running-time approximation in Table 5.3, which produce the results in Table 5.2.

Table 5.3: Hyper parameters and running-time approximation (minutes)

Method	Data set	Hyper parameters	Train	Test
Matrix Factorization	Algebra	β=0.005, #iter=120, K=16, λ=0.015	10.48	0.002
Biased Matrix Factorization	Algebra	β=0.002, #iter=80, K=64, λ=0.005	10.92	0.002
Student-kNN	Algebra	k=max, $\bar{k}$=1590.97, $simMeasure$=Cosine	8.51	5.68
Task-kNN	Algebra	k=max, $\bar{k}$=1755.82, $simMeasure$=Cosine	40.29	4.72
Matrix Factorization	Bridge	β=0.001, #iter=80, K=256, λ=0.0015	58.23	0.006
Biased Matrix Factorization	Bridge	β=0.001, #iter=80, K=128, λ=0.0015	36.93	0.004
Student-kNN	Bridge	k=max, $\bar{k}$=2314.99, $simMeasure$=Cosine	21.77	18.92
Task-kNN	Bridge	k=max, $\bar{k}$=1547.30, $simMeasure$=Cosine	46.67	8.98
Matrix Factorization	ASSISTments	β=0.01, #iter=60, K=32, λ=0.01	0.010	0.0009
Biased Matrix Factorization	ASSISTments	β=0.01, #iter=100, K=8, λ=0.05	0.013	0.0009
Student-kNN	ASSISTments	k=max, $\bar{k}$=2141.35, $simMeasure$=Cosine	1.85	5.20
Task-kNN	ASSISTments	k=max, $\bar{k}$=42.26, $simMeasure$=Cosine	0.079	0.05

k=max is using all nearest neighbors with $sim(s, s') > 0$ or $sim(i, i') > 0$;
$\bar{k}$ is the average of all "actual used" k-nearest neighbors

5.6 Discussion

Another important aspect of using the latent factor models is that besides predicting student performance, we can also use this approach for recommending the similar tasks (problems, exercises, etc) to the students and can determine which tasks are notoriously difficult for him/her since the predicted student performance represents the estimation of a student's performance on a given exercise. For example, there is a huge database of exercises where the students lose lots of time to solve the problems which are too easy or too hard for them. When a system is able to predict the student performance, it could recommend more appropriate tasks for them. Thus, we could filter out the tasks with predicted high performance (confidence) since these tasks are too easy for them, or filter out the tasks with predicted low performance (too hard for them) or both, depending on the goals of the e-learning system. We will discuss more about this problem in Chapter 9.

Moreover, an open issue is that each student performs many tasks, and each task relates to one or many required skills. Thus, using multi-relational matrix factorization (e.g., in (Lippert *et al.*, 2008; Singh & Gordon, 2008)) would also improve the prediction results. We will introduce this approach in the next chapter.

Chapter 6

Exploiting Multi-Relational Aspects in Student Modeling

Contents

6.1 Problem Reformulation for Multi-relational Data 58
6.2 Student Modeling with Multi-Relational Matrix Factorization (MRMF) 60
6.3 Student Modeling with Weighted Multi-Relational Matrix Factorization (WMRMF) 61
6.4 Experiments 63
6.4.1 Entity Relationship Diagram Revisions 63
6.4.2 Experimental Setting 63
6.4.3 Experimental Results 64
6.5 Conclusion 65

In the previous chapter we have shown that good results can be achieved by casting the predicting student performance (PSP) to rating prediction task in recommender systems and that matrix factorization (MF) is a promising approach for this problem. However, the previous state-of-the-art MF approaches for PSP only make use of one relationship, that is, between the students and the tasks or between the students and the skills needed to solve the tasks. In fact each student performs several tasks, and each task relates to the skill(s) needed to solve them, while the students are also required mastering on the skills that they have learned.

In this chapter we propose to exploit such multiple relationships for improving the prediction accuracy by using multi-relational matrix factorization (MRMF) methods. This approach has shown to be successful in recommender systems, e.g. in (Lippert

et al., 2008; Singh & Gordon, 2008), however, using it for student modelling (or educational data mining), especially in predicting student performance, for instance, is still a new topic. Moreover, we also propose a weighted multi-relational matrix factorization (WMRMF) to take into account the main relation which contains the target variable. We then evaluate the proposed methods on the large real-world data sets and compare their results with the other state-of-the-art methods in both recommender system and student modeling domains. We empirically show that by exploiting multiple relationships in student performance data, we can archive better prediction results.

6.1 Problem Reformulation for Multi-relational Data

In the previous chapter, we have used only a single relationship between the students and the tasks or a single relationship between the students and the skills which are required to solve the tasks. This can be represented in an entity relationship diagram (ERD), e.g., the relation "Performs" or the relation "Applies" in Figure 6.1, which can be represented as

$$\mathbf{R} = \{(S; I)\}$$

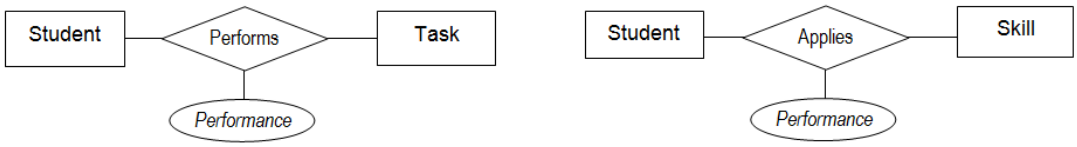


Figure 6.1: ERDs present using a single relation for matrix factorization

In this chapter, we also use the same formulation as in Chapter 2, however, we would like to exploit several possible relationships between the students, tasks, and their meta data, so the formulation needs to be extended. It is important to note that the extended formulation in this chapter is a generalization of the formulation in Chapter 2.

We denote

$$\{\mathbf{E}_1, \ldots, \mathbf{E}_N\}$$

as a set of N entity types (e.g. "Student", "Task", "Skill",...) and

$$\{\mathbf{R}_1, \ldots, \mathbf{R}_M\}$$

as a set of M binary relation types (e.g. "Performs", "Requires",...).

The problem now is to predict the values of the relation type between two entity types, for example,

$$\mathbf{R}_r = \{(E_{1_r}; E_{2_r})\} \quad (r = 1 \ldots M)$$

while taking into account the information in the other relations. Clearly, the multi-relational matrix factorization approach is a suitable choice for this problem.

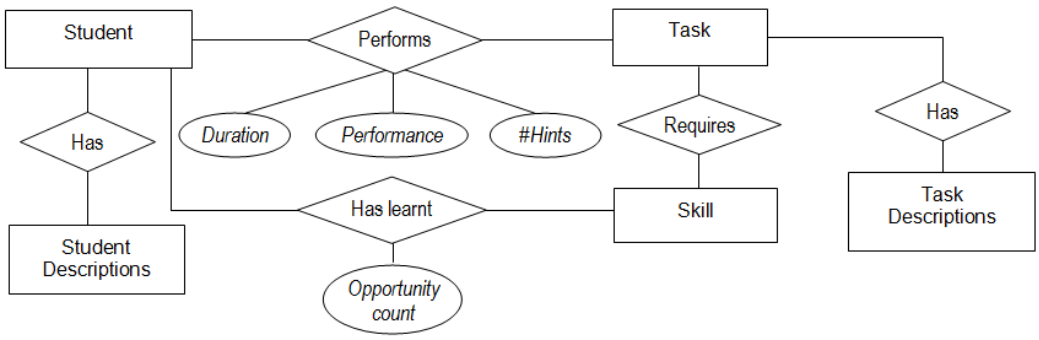


Figure 6.2: Entity relationship diagram includes useful information for PSP

Figure 6.2 presents an example of entity relationship diagram which covers the important information in predicting student performance. In this figure, each student performs the task and his/her performance is estimated by a performance score and a solving duration. The number of hints that the student requests are also expressed in this relationship. To solve the tasks correctly, the student needs to know specific skill(s), and the task itself also associates with the skill(s) that need to be learned by the students. The "opportunity count" attribute records how many times the student have chance to learn the skill. Moreover, each student (task) may be described by his/her (its) descriptions.

Figure 6.3 is an example of how to represent parts of the above ERD into matrices. The first matrix represents student performance on the given tasks (Student-Performs-Task relation); the second matrix represents whether the task requires the skills (Task-Requires-Skill relation); and the third matrix represents the number of opportunities that the student has encountered the skills (Student-HasLearnt-Skill relation).

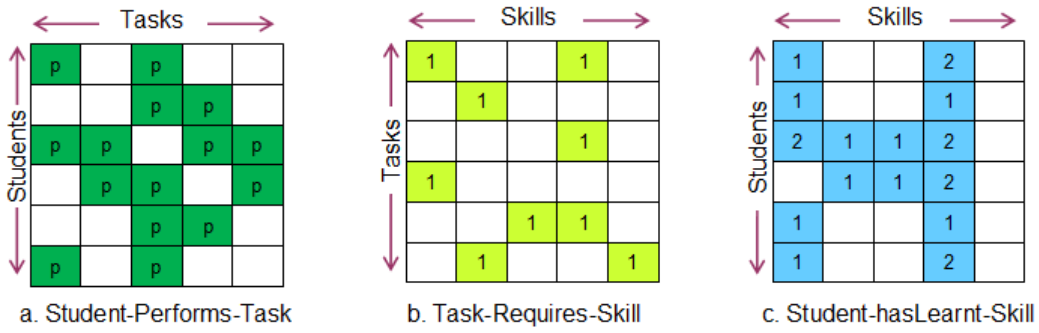


Figure 6.3: Examples of matrix representations (p is a performance score, e.g. $p \in [0..1]$)

6.2 Student Modeling with Multi-Relational Matrix Factorization (MRMF)

For purpose of extending to multi-relational case, we now review the Matrix Factorization method in slightly changed notations.

Matrix factorization is the task of approximating[1] a matrix $\mathbf{R} \in \mathbb{R}^{|S|\times|I|}$ by the product of two smaller matrices $\mathbf{W}_1$ and $\mathbf{W}_2$, i.e.,

$$\mathbf{R} \approx \mathbf{W}_1\mathbf{W}_2^T$$

$\mathbf{W}_1 \in \mathbb{R}^{|S|\times K}$ is a matrix where each row s is a vector containing the K latent factors describing the student s and $\mathbf{W}_2 \in \mathbb{R}^{|I|\times K}$ is a matrix where each row i is a vector containing the K latent factors describing the task i. Let $\mathbf{w}_{1_{sk}}$ and $\mathbf{w}_{2_{ik}}$ be the elements and $\mathbf{w}_{1_s}$ and $\mathbf{w}_{2_i}$ the vectors of $\mathbf{W}_1$ and $\mathbf{W}_2$, respectively, then the performance p given by student s to task i is predicted by:

$$\hat{p}_{si} = \sum_{k=1}^{K} \mathbf{w}_{1_{sk}}\mathbf{w}_{2_{ik}} = \mathbf{w}_{1_s}\mathbf{w}_{2_i}^T \tag{6.1}$$

$\mathbf{W}_1$ and $\mathbf{W}_2$ are the model parameters (latent factor matrices) which can be learned by optimizing the objective function (6.2) given a criterion, e.g. root mean squared error (RMSE), using stochastic gradient descent.

$$\mathcal{O}^{\text{MF}} = \sum_{(s,i)\in\mathbf{R}} \left((\mathbf{R})_{si} - \mathbf{w}_{1_s}\mathbf{w}_{2_i}^T\right)^2 + \lambda(||\mathbf{W}_1||_F^2 + ||\mathbf{W}_2||_F^2) \tag{6.2}$$

where $||\cdot||_F^2$ is a Frobenius norm and λ is a regularization term which is used to prevent over-fitting (please refer to the articles Koren (2010) for more details).

So far, we have briefly described the matrix factorization which uses only one relation type between two entity types (e.g. the relation "Performs" between "Student" and "Task" in Figure 6.2). The Multi-Relational Matrix Factorization (MRMF) (Lippert *et al.*, 2008; Singh & Gordon, 2008) is a general case of matrix factorization where we can include more than one relationship and more than two entity types.

Taking into account the multiple relationships between the entity types, the objective function of the MRMF is presented by:

$$\mathcal{O}^{\text{MRMF}} = \sum_{r=1}^{M} \sum_{(s,i)\in\mathbf{R}_r} \left((\mathbf{R}_r)_{si} - \mathbf{w}_{r_1 s}\mathbf{w}_{r_2 i}^T\right)^2 + \lambda \left(\sum_{j=1}^{N} ||\mathbf{W}_j||_F^2\right) \tag{6.3}$$

[1] It has been shown that this technique works well even when $\mathbf{R}$ is very sparse (Koren, 2010), which is usually the case in predicting student performance

where M is the number of relation types and $\{\mathbf{W}_j\}_{j=1...N}$ are the latent factor matrices of N entity types.

Please note that equation (6.3) is not the sum of independent terms. When learning the model parameters, every factor matrix is updated with respect to all relation types it involves until a common convergence is met (Lippert *et al.*, 2008) or the maximum number of predefined iterations is reached.

6.3 Student Modeling with Weighted Multi-Relational Matrix Factorization (WMRMF)

Using the MRMF, we can utilize many relationships between many entities. However, this method treats the important role of all relations equally. Clearly, we can see that the main relation which contains the target variable (e.g. "Student-Performs-Task" in Figure 6.2) is more important than the other supplement relations (e.g "Task-Requires-Skill"), thus it should have more weight.

We now propose the Weighted Multi-Relational Matrix Factorization (WMRMF) to take into account the importance of the main relation. So, the objective function in equation (6.3) now becomes:

$$\mathcal{O}^{\text{WMRMF}} = \sum_{r=1}^{M} \Theta_r \sum_{(s,i)\in\mathbf{R}_r} \left((\mathbf{R}_r)_{si} - \mathbf{w}_{r_1 s}\mathbf{w}_{r_2 i}^T\right)^2 + \lambda \left(\sum_{j=1}^{N} ||\mathbf{W}_j||_F^2\right) \tag{6.4}$$

where Θ_r is a weight function, for example, it sets the weight to maximum for the main relation and reduces the weight for the rest, as in equation (6.5). However, some other choices could also be considered.

$$\Theta_r = \begin{cases} 1, & \text{if } r \text{ is the main relation} \\ \theta, & \text{else } (0 < \theta \leq 1) \end{cases} \tag{6.5}$$

where θ is the weight hyper parameter which can be determined from the training data. Another important property of the WMRMF is that in an extreme case ($\theta = 1$), the WMRMF is equivalent to the MRMF.

The WMRMF updates its latent factors for each relation at iteration n via equations (6.6) and (6.7):

$$\mathbf{w}_{r_1 s}^n = \mathbf{w}_{r_1 s}^{n-1} - \beta \left(\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_1 s}^{n-1}}\right) \tag{6.6}$$

$$\mathbf{w}_{r_2 i}^n = \mathbf{w}_{r_2 i}^{n-1} - \beta \left(\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_2 i}^{n-1}}\right) \tag{6.7}$$

where β is a learning rate; and the gradients $\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_1 s}}$ and $\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_2 i}}$ are determined by

$$\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_1 s}} = \lambda \mathbf{w}_{r_1 s} - 2\Theta_r \left((\mathbf{R}_r)_{si} - \mathbf{w}_{r_1 s} \mathbf{w}_{r_2 i}^T \right) \mathbf{w}_{r_2 i} \tag{6.8}$$

$$\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_2 i}} = \lambda \mathbf{w}_{r_2 i} - 2\Theta_r \left((\mathbf{R}_r)_{si} - \mathbf{w}_{r_1 s} \mathbf{w}_{r_2 i}^T \right) \mathbf{w}_{r_1 s} \tag{6.9}$$

The WMRMF's learning process is summarized in Algorithm 4. We initialize the latent factor matrices from the normal distribution $\mathcal{N}(\mu, \sigma^2)$, e.g. mean $\mu = 0$ and standard deviation $\sigma^2 = 0.01$, and initialize the weight value for each relation types using equation 6.5. While the stopping condition is not met, e.g. reaching the maximum number of iterations or converging ($\mathcal{O}_{Iter_{(n-1)}}^{\text{WMRMF}} - \mathcal{O}_{Iter_n}^{\text{WMRMF}} < \epsilon$), the latent factors are updated iteratively.

Algorithm 4 Learn the WMRMF for $\mathbf{E}_1, \ldots, \mathbf{E}_N$ entity types and $\mathbf{R}_1, \ldots, \mathbf{R}_M$ relation types; λ is the regularization term, β is the learning rate, K is the number of latent factors, θ is the weight value, and a stopping criterion.

procedure LEARNWMRMF($\mathbf{E}_1, \ldots, \mathbf{E}_N$; $\mathbf{R}_1, \ldots, \mathbf{R}_M$; λ; β; K; θ; stopping criterion)
  for $j \leftarrow 1 \ldots N$ **do**
    $\mathbf{W}_j \leftarrow$ Draw randomly from $\mathcal{N}(\mu, \sigma^2)$
  end for
  for $r \leftarrow 1 \ldots M$ **do**
    Initialize Θ_r using equation (6.5)
  end for
  while (Stopping criterion is NOT met) **do**
    for each relation $\mathbf{R}_r = \{(E_{1_r}; E_{2_r})\}$ in $\{\mathbf{R}_1, \ldots, \mathbf{R}_M\}$ **do**
      for $l \leftarrow 1 \ldots |\mathbf{R}_r|$, **do**
        Draw randomly (s, i) in $\mathbf{R}_r$

        $\mathbf{w}_{r_1 s} \leftarrow \mathbf{w}_{r_1 s} - \beta \left(\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_1 s}} \right)$

        $\mathbf{w}_{r_2 i} \leftarrow \mathbf{w}_{r_2 i} - \beta \left(\frac{\partial \mathcal{O}_{si}^{\text{WMRMF}}}{\partial \mathbf{w}_{r_2 i}} \right)$

      end for
    end for
  end while
  return $\{\mathbf{W}_j\}_{j=1 \ldots N}$
end procedure

After the learning process, the model parameters $\{\mathbf{W}_j\}_{j=1...N}$ are obtained, then we can generate the prediction for any relation using the same equation (6.1).

6.4 Experiments

In this section, we first revise the entity relationship diagrams to adapt with the available real-world data sets. We then describe the experimental setting, and finally the experimental results including the comparison with other state-of-the-art methods are presented.

6.4.1 Entity Relationship Diagram Revisions

In the specific data sets used for experiments, several information, e.g. "#Hints" and "Durations" (start time, end time), are not provided in the test sets (the KDD Cup 2010 data). Thus, for applying the MRMF and WMRMF, the ERD in Figure 6.2 needs to be narrowed down.

We propose two different ERDs for experiments as in Figure 6.4. In each ERD, we also present which relation can be used as the main relation (filled by gray color), which has higher weight for the WMRMF.

Moreover, the relation "Has learnt" in Figure 6.2 is also revised. Instead of using "opportunity counts" as the values for this relation, we use "average performance" of the student on the skill. By this way, we can also predict "how the student master on the given skill".

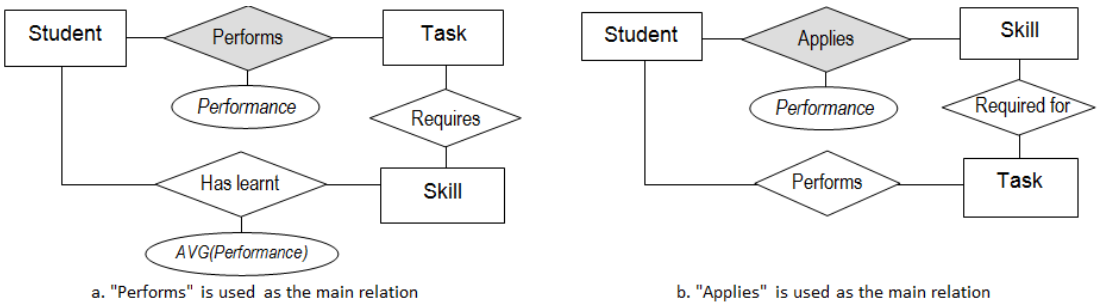


Figure 6.4: Entity relationship diagrams are used for experiments

Tables 6.1 and 6.2 summarize the information about the number of entities and relations in the above Figure, for three data sets which were described in Chapter 3.

6.4.2 Experimental Setting

We use the same settings as described in Section 5.5.2. For the weight θ in the WMRMF, due to large number of hyper parameter spaces we have also done a raw search,

Table 6.1: Entity information from three data sets

Entity	Dataset	#Entities	Dataset	#Entities	Data set	#Entities
Student	Algebra	3,310	Bridge	6,043	ASSISTments	8,519
Task	Algebra	1,416,473	Bridge	887,740	ASSISTments	35,978
Skill	Algebra	2,979	Bridge	1,458	ASSISTments	348

Table 6.2: Relation information from three data sets

Relation	Dataset	#Rows	Dataset	#Rows	Data set	#Rows
Performs	Algebra	8,918,054	Bridge	20,012,498	ASSISTments	829,671
Requires	Algebra	1,961,566	Bridge	1,512,280	ASSISTments	30,904
HasLearnt	Algebra	1,707,794	Bridge	2,619,400	ASSISTments	110,321

Note: Relation "Performs" has the same information with "Applies", and Relation "Requires" has the same information with "Required for".

e.g., finding $\theta \in (0.7, 0.75, 0.8, 0.85)$.

6.4.3 Experimental Results

Figure 6.5 presents the RMSE results of the proposed methods and the other baselines (using "Student-Performs-Task" as the main relation, presented in Figure 6.4a).

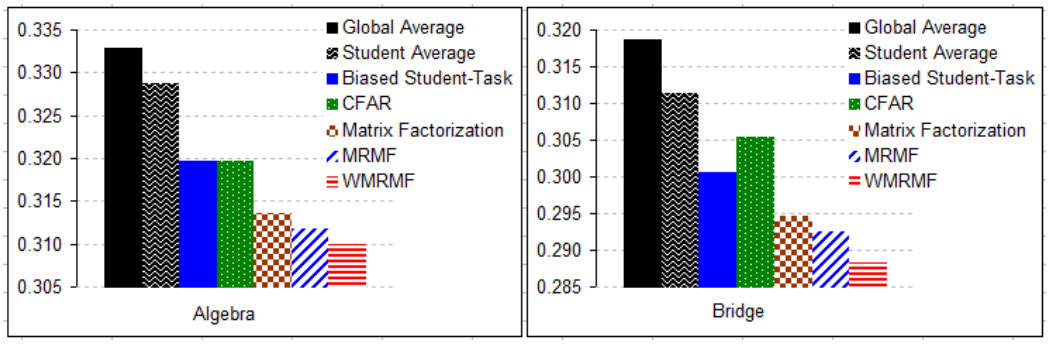


Figure 6.5: Using Student-Performs-Task as the main relation: RMSE of (W)MRMF vs. the other models

The MRMF and WMRMF, which take into account the multiple relationships between entities, have improvements compared to the others. These results also consist with previous work, e.g in Lippert *et al.* (2008), which shown that the multi-relational approach can improve over the single relational matrix factorization.

From Figure 6.5, we can also observe that the factorization models perform better on the Bridge data set. The reason could be because the Bridge data set is less sparse than the Algebra data set (on average, the Bridge data set has 22.52 performances/task (ratings/item), while the Algebra has 6.27 performances/task).

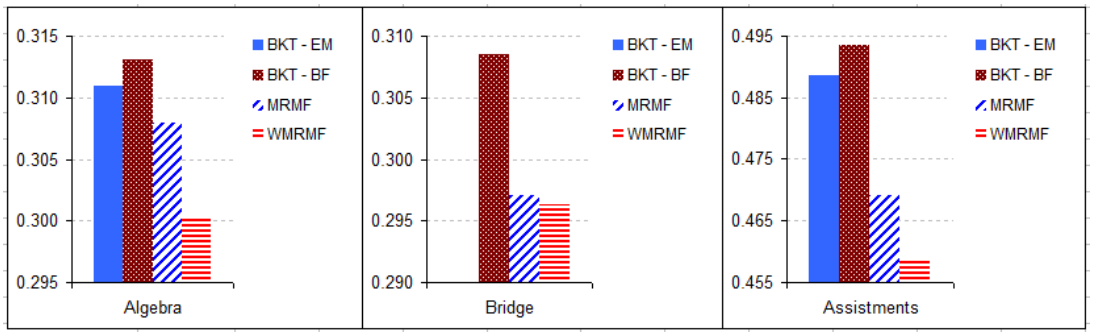


Figure 6.6: Using Student-Applies-Skill as the main relation: RMSE of (W)MRMF vs. the state-of-the-art BKT models

Figure 6.6 presents the RMSE results of using "Student-Applies-Skill" as the main relation (presented in Figure 6.4b). In this case, instead of predicting student performance on particular task directly, we predict the student performance on the required skills associated with the task. This has been done in the literature, e.g. in Baker *et al.* (2008b); Chang *et al.* (2006); Corbett & Anderson (1995), for student modeling to trace how students apply their gained knowledge/skill on the given tasks. Thus, we have also experimented both the BKT-BF (Baker *et al.*, 2008b) and the BKT-EM (Chang *et al.*, 2006) on these data to understand how far the proposed models are improved. However, it is quite expensive to obtain the BKTs' hyper parameters using the Brute-Force method.

Clearly, from the results of Figure 6.6 we can see that the RMSE of the proposed methods also have improvements compared to the state-of-the-art BKT-BF. The BKT-EM were intractable on Bridge data set, so we have not reported this result.

For referencing, we report the hyper parameters found in Table 6.3. Running time of the WMRMF using these hyper parameters is ≈6.0 hours on the largest data set (Bridge), however, in educational environment where the models need not to be re-trained continuously, this running time would not be an issue.

6.5 Conclusion

In this chapter, we have proposed a novel approach which uses multi-relational matrix factorization (MRMF) to exploit the relationships between students, tasks, and other meta data in predicting student performance. We have also proposed a weighted MRMF (WMRMF) to take into account the main relation that contains the target variable. Furthermore, we have shown how to present the relationships of the student performance data to multiple matrices and validate the proposed approach using three large data sets. Experimental results show that this approach can perform nicely compared to the other methods.

However, incorporating specific latent factors for the entities, e.g. as described in

Table 6.3: Hyper parameters are used for experiments.

Method	Data set	Hyper parameters
MF	Algebra	β=0.005, #iter=120, K=16, λ=0.015
MRMF	Algebra	β=0.0005, #iter=1000, K=16, λ=0.00055
WMRMF	Algebra	β=0.001, #iter=550, K=16, λ=0.00125, θ=0.85
MF	Bridge	β=0.01, #iter=80, K=64, λ=0.015
MRMF	Bridge	β=0.0005, #iter=700, K=40, λ=0.00055
WMRMF	Bridge	β=0.001, #iter=550, K=80, λ=0.001, θ=0.7
MF	ASSISTments	β=0.01, #iter=80, K=64, λ=0.015
MRMF	ASSISTments	β=0.005, #iter=20, K=64, λ=0.015
WMRMF	ASSISTments	β=0.0015, #iter=60, K=16, λ=0.005, θ=0.7

Note: λ is the regularization term, β is the learning rate, K is the number of latent factors, θ is the weight value and #iter is the number of iterations.

Töscher & Jahrer (2010), and combining the results of different parameters using ensemble methods may produce better results. Moreover, adding more data relationships to the models (if applicable, e.g., the durations of performing the tasks, which highly reflect the task's difficulty; the number of hints that the student requested;...) may also lead to further improvement.

Chapter 7

Personalized Forecasting Student Performance

Contents

7.1 Classical Forecasting Techniques 68
7.2 Personalized Forecasting Techniques 70
7.2.1 Strategies for Using Historical Data 70
7.2.2 Non-Biased Personalized Forecasting 72
7.2.3 Personalized Forecasting with "Student-Task-Biases" 74
7.2.4 Personalized Forecasting with "Discounted-Mean" 76
7.3 Bootstrapping and k-Step-Ahead Forecasting 77
7.4 Experiments 77
7.4.1 Experimental Setting 78
7.4.2 Experimental Results 79
7.5 Conclusions 81

In Chapter 5, we have proposed using the latent factor models for predicting student performance. These models might implicitly encode the student/task latent factors and the student/task effects. However, these models have not taken the temporal/sequential effect into account.

Obviously, from the educational point of view, the students' knowledge cumulates and improves over time. For example, the second time a student is doing his/her exercises, his/her performance on average gets better. Another fact is that "the more the students (learners) study, the better the performance they get". Therefore, the sequential/temporal effect is an important aspect for predicting student performance.

Moreover, as a consequence from the knowledge acquisition process, there may have a trend in the student performance. Similar to other domains, where the target

distributions show the trend and the temporal/sequential effect needs to be taken into account, forecasting techniques are reasonable choices.

In this chapter, we propose *personalized forecasting* methods for student modeling, especially for forecasting/predicting student performance. Different from the other literature in predicting student performance, e.g., in Romero *et al.* (2008); Shen *et al.* (2010); Yu *et al.* (2010) etc., where all historical data are used to form the prediction models, our approach only uses historical information of individual student to forecast/predict his/her own performance. Moreover, instead of using prediction, we use the forecasting techniques to better capture the temporal/sequential effect. The proposed personalized forecasting techniques also incorporate the "student effect" and the "task effect" into their models.

In the next sections, we first introduce the standard forecasting techniques such as single exponential smoothing and double exponential smoothing forecaster (Box & Jenkins, 1990; Brockwell & Davis, 2002). We then propose the personalized forecasting approaches for the student performance prediction problem. Next, we show how to use historical data in an efficient way and we analyze the data in order to discover how the temporal/sequential aspect affects on the student performance. Finally, the experiments are conducted to validate the proposed personalized forecasting methods as well as to compare with the other state-of-the-art methods.

7.1 Classical Forecasting Techniques

One of the simplest forecasting techniques is Moving Average . Given a sequence of observations $p_1, p_2, \ldots, p_{t-1}$ and a time period L, then the forecasting value at time t using the moving average method is determined by

$$\hat{p}_t = \frac{1}{L}\sum_{l=t-L}^{t} p_l = \frac{1}{L}\cdot p_{t-L+1} + \frac{1}{L}\cdot p_{t-L+2} + \ldots + \frac{1}{L}\cdot p_{t-1} \tag{7.1}$$

In the moving average method, the past observations are weighted equally (i.e., the weights assigned to the past observations are equal to $1/L$). However, in several cases of real applications, the recent data are considered more important than the old data, thus, recent observations are given relatively more weight in forecasting than the older observations (NIST, 2010). In those cases, exponential smoothing techniques would be better choices (Brockwell & Davis, 2002).

Using single exponential smoothing, the forecasting value at time t, which is depicted in Figure 7.1, is determined by:

$$\hat{p}_t = E_t \qquad (t > 2) \tag{7.2}$$

where E_t is the smoothing value at time t which is determined by

$$E_t = \alpha p_{t-1} + (1-\alpha)E_{t-1} \qquad (0 < \alpha < 1) \tag{7.3}$$

where p_{t-1} and E_{t-1} are the actual value and smoothing value at time $t-1$, respectively; and α is a smoothing constant. For the initializing values, there is no E_1, and E_2 can be set to the first observation, e.g. $E_2 = p_1$.

We will call this method **SEF** (Single Exponential Smoothing Forecaster).

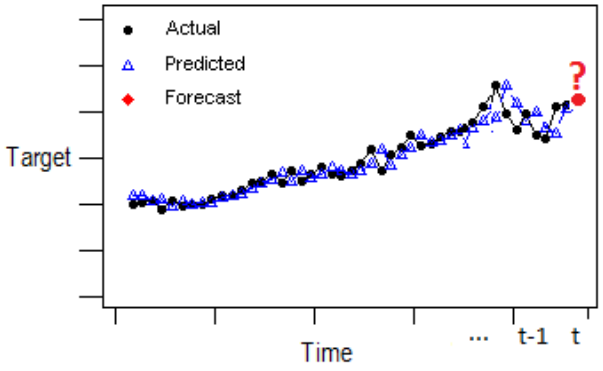


Figure 7.1: An illustration of forecasting at time t

Moreover, when the observations show the trend, the better forecasting results can be archived by using double exponential smoothing (please note that the other literature also call this method the "Holt forecasting" (Brockwell & Davis, 2002; Chatfield & Yar, 1988)).

Using double exponential smoothing technique, the value at time t is forecasted by

$$\hat{p}_t = E_t + T_t \qquad (t > 2) \tag{7.4}$$

where E_t is the smoothing value at time t which is determined by

$$E_t = \alpha p_{t-1} + (1-\alpha)(E_{t-1} + T_{t-1}) \qquad (0 < \alpha < 1) \tag{7.5}$$

T_t is the trend value at time t which is determined by

$$T_t = \beta(E_t - E_{t-1}) + (1-\beta)T_{t-1} \qquad (0 < \beta < 1) \tag{7.6}$$

and β is the trend constant.

The initial values can be set as $E_2 = p_1$ and $T_2 = p_2 - p_1$. However, many other initializing strategies can also be used (see Box & Jenkins (1990); Brockwell & Davis (2002) for details).

We will call this method **DEF** (Double Exponential Smoothing Forecaster).

7.2 Personalized Forecasting Techniques

Traditionally, to solve a prediction (classification/regression) problem (the predicting student performance in our case) we usually collect all the observed data and build a prediction model, e.g. using Support Vector Machines, Logistic Regression etc, on those data, and finally, using that model to predict the new data.

However, the proposed method differs from the above approach, instead of using all historical data to form the models, we only use the historical data of individual student to build the model to forecast/predict his/her own performance. Here is an example including a picture to describe the reason for our choice:

> *"Tom is a clever student (a person with a book at hand in Figure 7.2). His performance is always better/higher than the other students, while Bob is a student who has a medium-performance (a person with a football at hand in Figure 7.2). Obviously, using Bob's performance information to predict/forecast Tom's would not fit*[1]*. We should use either the information of "similar students" to predict the other ones by using collaborative data as described in Chapter 5 or we should use one's performance to predict himself/herself as the proposed personalized forecasting methods in this chapter".*

7.2.1 Strategies for Using Historical Data

We propose two approach for using the historical data to form the personalized forecasting models.

The first approach is using *history length* L to control the length of the historical data. For example, L could be how many previous solving-steps which we can use to forecast the next solving-steps. We call this method *"histLength"*.

In the second approach, instead of using the history length L, we can use all historical data in the same *unit* and *section* to forecast the performance of new problems in that unit and section. We call this approach *"secLength"*.

An illustration of the *histLength* L and *secLength* is presented in Figure 7.3. For examples, to forecast Tom's performance on the steps of the "Problem 3", we can use all previous steps with the length L including the steps of other previous units and sections, or we can use the *secLength* which includes the previous steps in the same "Unit 5" and "Section 5" (e.g., the steps of "Problem 1" and "Problem 2") to form the models.

[1]It is important to note that in Chapter 5 we have proposed using collaborative data (co-relations between students/tasks) for predicting student performance. In that case, we have predicted the performance of an individual student by using only the performance of the "similar students" (but not all of the other ones); while in this chapter we will propose using the performance of an individual student to predict himself/herself. Obviously, these two approaches are not contradictory.

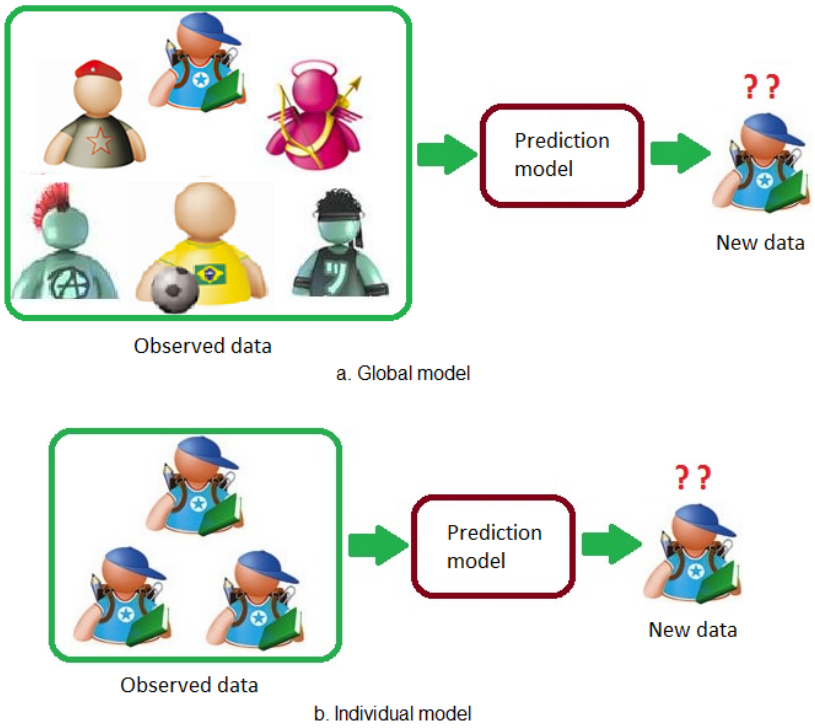


Figure 7.2: An example of using personalization in PSP (*global model*: Using information (performances) of all other students to predict / forecast Tom's performance; and *individual model*: Using Tom's performance to predict / forecast himself)

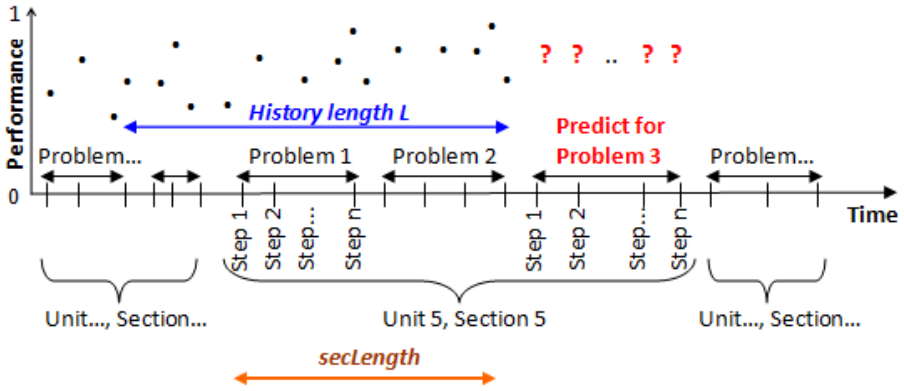


Figure 7.3: An illustration of personalized forecasting which uses all historical data controlled by the history length L, or using all historical data in the same unit and section. This illustration describes for one student.

7.2.2 Non-Biased Personalized Forecasting

In this approach, we take into account the individualization but without using the student/task biases.

The personalized forecasting for student s at time t using single exponential smoothing is determined by:

$$\hat{p}_t^s = E_t^s \qquad (t > 2) \tag{7.7}$$

where E_t^s is the smoothing value at time t for student s, which is determined by

$$E_t^s = \alpha \cdot p_{t-1}^s + (1-\alpha) \cdot E_{t-1}^s \qquad (0 < \alpha < 1) \tag{7.8}$$

where p_{t-1}^s and E_{t-1}^s are the actual performance (observation) and the smoothing value of student s at time $t-1$, respectively, and α is a smoothing parameter. Similar to traditional forecasting method, we also need to initialize the value for the E_2^s, e.g., $E_2^s = p_1^s$. We call this method **N-PSEF** (Non-biased Personalized Single Exponential smoothing Forecaster).

In this approach, the smoothing parameter α can be learned by either using grid-search (brute-force) or using stochastic gradient descent, which is denoted as **N-PSEF-SGD**.

In case of using gradient descent, we would like to find the optimal α for a given criterion such as the sum squared error (or RMSE). The objective function which we want to minimize can be written as

$$\mathcal{O}(\alpha) = \sum_t (p_t^s - \hat{p}_t^s)^2 \tag{7.9}$$

By expansion using equation 7.8, the $\hat{p}_t$ can be written as

$$\hat{p}_t^s = \alpha \cdot p_{t-1}^s + (1-\alpha) \cdot \hat{p}_{t-1}^s \tag{7.10}$$

$$= \alpha \cdot p_{t-1}^s + (1-\alpha) \cdot \alpha \cdot p_{t-2}^s + (1-\alpha)^2 \cdot \hat{p}_{t-2}^s$$

$$\dots$$

$$= (1-\alpha)^{L-1} \cdot p_{t-L}^s + \alpha \cdot \sum_{j=1}^{L-1} (1-\alpha)^{j-1} \cdot p_{t-j}^s$$

where L is the history length as already mentioned in Section 7.2.1.

The gradient for updating the α can be computed as

$$\frac{\partial \mathcal{O}(\alpha)}{\partial \alpha} = -2 \cdot (p_t^s - \hat{p}_t^s) \cdot \frac{\partial \hat{p}_t^s}{\partial \alpha}$$

where

$$\frac{\partial \hat{p}_t^s}{\partial \alpha} = \frac{\partial \left((1-\alpha)^{L-1} \cdot p_{t-L}^s + \alpha \cdot \sum_{j=1}^{L-1} (1-\alpha)^{j-1} \cdot p_{t-j}^s \right)}{\partial \alpha}$$

$$= -(L-1)(1-\alpha)^{L-2} \cdot p_{t-L}^s + \sum_{j=1}^{L-1} (1-\alpha)^{j-2} \cdot (1-2\alpha) \cdot p_{t-j}^s \tag{7.11}$$

Using above gradient, the value of the α can be updated by

$$\alpha \leftarrow \alpha - \gamma \cdot \frac{\partial \mathcal{O}(\alpha)}{\partial \alpha} \tag{7.12}$$

where γ is a learning rate.

Algorithm 5 Learn the personalized single exponential smoothing forecaster (without using biases) using stochastic gradient descent, with learning rate γ, history length L, and a stopping criterion.

procedure N-PSEF-SGD($\mathcal{D}^{train}$; γ; L; stopping criterion)
  for $s \in S$ **do**
    $\alpha_s \leftarrow$ Draw randomly from $\mathcal{N}(\mu, \sigma^2)$
  end for
  while (stopping criterion is NOT met) **do**
    Draw randomly (s, i, p) from $\mathcal{D}^{train}$ at row T (considered as time T)
    $E_0^s \leftarrow p_{T-L}$
    for $t \leftarrow 1, \cdots, L-1$ **do**
      $E_t^s = \alpha_s \cdot p_{T-L+t}^s + (1-\alpha_s) \cdot E_{t-1}^s$
    end for
    $\hat{p}_T^s \leftarrow E_{L-1}^s$
    $\alpha_s \leftarrow \alpha_s - \gamma \cdot \frac{\partial \mathcal{O}(\alpha_s)}{\partial \alpha_s}$
  end while
  return { α }
end procedure

The learning algorithm for the **N-PSEF-SGD** is summarized in Algorithm 5. First, we initialize α from normal distribution $\mathcal{N}(\mu, \sigma^2)$, e.g., mean $\mu = 0.2$ and standard deviation $\sigma^2 = 0.1$. While the stopping condition is not met, e.g., neither reaching the

maximum number of iterations nor converging ($\mathcal{O}(\alpha)_{Iter_{(n-1)}} - \mathcal{O}(\alpha)_{Iter_n} < \epsilon$), the α values are updated iteratively.

Moreover, if we would like to take the trend into account, the double exponential smoothing forecasting should be used. With this, the performance of student s at time t is now forecasted by:

$$\hat{p}_t^s = E_t^s + T_t^s \tag{7.13}$$

where E_t^s is now determined by

$$E_t^s = \alpha p_{t-1}^s + (1-\alpha)(E_{t-1}^s + T_{t-1}^s) \qquad (0 < \alpha < 1) \tag{7.14}$$

and T_t^s is now determined by

$$T_t^s = \beta(E_t^s - E_{t-1}^s) + (1-\beta)T_{t-1}^s \qquad (0 < \beta < 1) \tag{7.15}$$

For the initialized values, there is no E_1^s, and we can also initialize $E_2^s = p_1^s$ and $T_2^s = p_2^s - p_1^s$. The α and β parameters can be learned in a similar way described in previous section. We call this method **N-PDEF** (Non-biased Personalized Double Exponential smoothing Forecaster).

7.2.3 Personalized Forecasting with "Student-Task-Biases"

In this approach, we again adopt the idea in recommender systems to employ the "student bias" and "task bias" to the models, as described in Section 5.3. Using these biased terms, the performance of student s for task i at time t is now forecasted by

$$\hat{p}_t^s = \delta \cdot \hat{p}^{b_{si}} + (1-\delta) \cdot E_t^s \tag{7.16}$$

where δ acts as a global weight for "making ensemble" between the biased predictor $\hat{p}^{b_{si}}$ and the smoothing value E_t^s which can be obtained from the single exponential smoothing (equation 7.8) or double exponential smoothing (equation 7.14), and $\hat{p}^{b_{si}}$ is determined by

$$\hat{p}^{b_{si}} = \mu + b_s + b_i \tag{7.17}$$

where μ, b_s, and b_i are the global average, student bias, and task bias, as already introduced in Section 5.3, respectively.

We call these methods **B-PSEF** (Biased Personalized Single Exponential smoothing Forecaster) and **B-PDEF** (Biased Personalized Double Exponential smoothing Forecaster).

Algorithm 6 Learn the biased personalized single exponential smoothing forecaster using stochastic gradient descent, with learning rate γ, global weight δ, history length L, and a stopping criterion.

procedure B-PSEF-SGD($\mathcal{D}^{train}$; γ; δ; L; stopping criterion)
  for $s \in S$ **do**
    $\alpha_s \leftarrow$ Draw randomly from $\mathcal{N}(0.2, \sigma^2)$
  end for
  $\mu \leftarrow \frac{\sum_{p \in \mathcal{D}^{train}} p}{|\mathcal{D}^{train}|}$
  for each student s **do**
    $b_s \leftarrow \frac{\sum_i (p_{si} - \mu)}{|\mathcal{D}_s^{train}|}$
  end for
  for each task i **do**
    $b_i \leftarrow \frac{\sum_s (p_{si} - \mu)}{|\mathcal{D}_i^{train}|}$
  end for
  while (stopping criterion is NOT met) **do**
    Draw randomly (s, i, p) from $\mathcal{D}^{train}$ at row T (considered as time T)
    $E_0^s \leftarrow p_{T-L}$
    for $t \leftarrow 1, \cdots, L-1$ **do**
      $E_t^s = \alpha_s \cdot p_{T-L+t}^s + (1 - \alpha_s) \cdot E_{t-1}^s$
    end for
    $\hat{p}^{b_{si}} \leftarrow \mu + b_s + b_i$
    $\hat{p}_T^s \leftarrow \delta \cdot \hat{p}^{b_{si}} + (1 - \delta) \cdot E_{L-1}^s$
    $e_T \leftarrow p_T^s - \hat{p}_T^s$
    $b_s \leftarrow b_s + \gamma \cdot e_T$
    $b_i \leftarrow b_i + \gamma \cdot e_T$
    $\alpha_s \leftarrow \alpha_s - \gamma \cdot \frac{\partial \mathcal{O}^b(\alpha_s)}{\partial \alpha_s}$
  end while
  return { α, b_s, b_i }
end procedure

Similar to the N-PSEF in previous section, parameters of the B-PSEF can also be learned by using either grid-search or using gradient descent, which is denoted as **B-PSEF-SGD**. The objective function can be written as

$$\mathcal{O}^b(\alpha) = \sum_t (p_t^s - \delta \cdot \hat{p}^{b_{si}} - (1 - \delta) \cdot \hat{p}_t^s)^2 \tag{7.18}$$

The gradient $\frac{\partial \mathcal{O}^b(\alpha)}{\partial \alpha}$ is calculated by

$$\frac{\partial \mathcal{O}^b(\alpha)}{\partial \alpha} = -2 \cdot (p_t^s - \delta \cdot \hat{p}^{b_{si}} - (1 - \delta) \cdot \hat{p}_t^s) \cdot (1 - \delta) \cdot \frac{\partial \hat{p}_t^s}{\partial \alpha} \tag{7.19}$$

where $\frac{\partial \hat{p}_t^s}{\partial \alpha}$ is determined by using equation 7.11.

The learning algorithm for the **B-PSEF-SGD** is summarized in Algorithm 6. First, we initialize α from normal distribution $\mathcal{N}(\mu, \sigma^2)$, e.g., mean $\mu = 0.2$ and standard deviation $\sigma^2 = 0.1$. Then, we compute the global average and the biased terms. While the stopping condition is not met, e.g., neither reaching the maximum number of iterations nor converging ($\mathcal{O}^b(\alpha)_{Iter_{(n-1)}} - \mathcal{O}^b(\alpha)_{Iter_n} < \epsilon$), the α values are updated iteratively.

7.2.4 Personalized Forecasting with "Discounted-Mean"

The true fact is that "human's memory is limited", thus, the students might forget what they have studied in the past, e.g., they might perform better on lessons they have learned recently than on such they have learned last year or before. Moreover, as already mentioned, from the educational point of view: "The more the learners study the better the performance they get" and "student's knowledge cumulate and improve over time", therefore, the sequential effect is an important factor for predicting student performance.

Instead of using smoothing values as in the previous sections, we now use a discounted mean value Θ, which reduces the weight controlled by a parameter α when going back to the history, e.g., Θ is calculated by

$$\Theta = \frac{\sum_{j=1}^{L} p_{t-j}^{s} \cdot e^{-\alpha \cdot j}}{\sum_{j=1}^{L} e^{-\alpha \cdot j}} \qquad (0 < \alpha < 1) \tag{7.20}$$

Using this discounted mean value, the performance of student s at time t is forecasted by

$$\hat{p}_t^s = \delta \cdot \hat{p}^{b_{si}} + (1 - \delta) \cdot \Theta \tag{7.21}$$

Similar to Section 7.2.3, the δ acts as a global weight for "making ensemble" between the biased predictor $\hat{p}^{b_{si}}$ and the "discounted-mean" Θ. The α parameter can be learned by the same way as in previous sections. We call this method **B-PDMF** (Biased Personalized Discounted-Mean Forecaster).

The B-PDMF has slightly different from the B-PSEF. For instance, the data weight is damped by a factor of $\alpha \cdot (1-\alpha)^t$ in B-PSEF, while the data is weighted by $\frac{e^{-\alpha \cdot t}}{\sum_{j=1}^{L} e^{-\alpha \cdot j}}$ in B-PDMF. Indeed, for analyzing the difference between these two methods (B-PSEF and B-PDMF), we rewrite their prediction functions in an expansion way as in the following. (for distinguished purpose, we change the α in B-PSEF to α')

$$\text{B-PSEF:} \quad \hat{p}_t^s = \underbrace{(1-\alpha')^{L-1}}_{W_L} \cdot p_{t-L}^s + \sum_{j=1}^{L-1} \underbrace{\alpha' \cdot (1-\alpha')^{j-1}}_{W_j} \cdot p_{t-j}^s$$

$$\text{B-PDMF: } \hat{p}_t^s = \underbrace{\frac{e^{-\alpha \cdot L}}{\sum_{j=1}^{L} e^{-\alpha \cdot j}}}_{W_L} \cdot p_{t-L}^s \quad + \quad \sum_{j=1}^{L-1} \underbrace{\frac{e^{-\alpha \cdot j}}{\sum_{j=1}^{L} e^{-\alpha \cdot j}}}_{W_j} \cdot p_{t-j}^s$$

The weights of B-PSEF and B-PDMF are summarized in Table 7.1. Thus, if we set $\alpha' = 1 - e^{-\alpha}$, both methods have the same weights when $j < L - 1$, however, they are different at the point $L - 1$.

Table 7.1: Comparison of two weighting approaches

Weight		B-PSEF	B-PDMF
$\frac{W_j}{W_{j+1}}$	$(j < L-1)$	$1 - \alpha'$	$e^{-\alpha}$
$\frac{W_{L-1}}{W_L}$	$(j = L-1)$	$\frac{\alpha'}{1-\alpha'}$	$e^{-\alpha}$

7.3 Bootstrapping and k-Step-Ahead Forecasting

One problem with the forecasting techniques is that what happens if we wish to forecast from some origins, usually the last data points, and there are no actual observations? For example, if we would like to forecast the "Step 4" of "Problem 3" in Figure 7.3, and there are no observations from "Step 1" to "Step 3" of that problem.

To solve this, we can use one of the approaches called *bootstrapping* (Brockwell & Davis, 2002; NIST, 2010). For example, using bootstrapping, the smoothing value in the equation 7.8 of the **N-PSEF** will become

$$E_t^s = \alpha \; p_{origin}^s + (1 - \alpha) E_{t-1}^s \tag{7.22}$$

where p_{origin}^s is a constant, e.g., the last actual observation. For example, the last step of the Problem 2 in Figure 7.3.

Another approach we can use is *k-step-ahead* forecasting. For example, using the k-step-ahead forecasting, the function in the equation 7.13 of the **N-PDEF** will become

$$\hat{p}_t^s = E_t^s + kT_t^s \tag{7.23}$$

where k is a constant which determines how many data points (steps) we would like to forecast-ahead.

7.4 Experiments

For evaluating the proposed methods, we also use the same metric with previous chapters - the root mean squared error (RMSE), and getting the RMSE score from the real system (on the KDD Cup 2010 website).

7.4.1 Experimental Setting

Initializations

We can initialize the smoothing and the trend values as $E_2^s = p_1^s$ and $T_2^s = p_2^s - p_1^s$, where p_1^s is the first actual performance of student s in a sequence which has length L.

However, from the preliminary results, we found that the student performance has fluctuations (we will see in Figure 7.5), so we initialize the smoothing values by averaging the previous performances:

$$E_2^s = \frac{\sum_{t=T-L}^{T-1} p_t^s}{L} \tag{7.24}$$

where T is the current time. In preliminary experiments, this initialization empirically works better than initializing with $E_2^s = p_1^s$.

Bootstrapping

Instead of using bootstrapping or k-step-ahead forecasting, in this chapter, from preliminary results we recognized that forecasting based on the previous forecast-values can perform better than using bootstrapping or k-step-ahead. For example, after having the forecast value of "Step 1", we use it as an observation for forecasting the "Step 2", and so on.

Baselines

The proposed approaches were compared with the standard single/double exponential forecasting (SEF/DEF) (Brockwell & Davis, 2002), with the *global average*, *student average* (user average), *biased-student-task* (this method is adapted from the user-item-baseline in Koren (2010)), and with the CFAR (Yu *et al.*, 2010). We also compare the personalized forecasting with the *matrix factorization* and logistic regression as described in previous chapters..

Furthermore, the proposed methods are compared with the state-of-the-art methods in student modeling: The Bayesian Knowledge Tracing using Brute Force (BKT-BF) (Baker *et al.*, 2008b; Corbett & Anderson, 1995) and the Bayesian Knowledge Tracing using Expectation Maximization (BKT-EM) (Chang *et al.*, 2006; Corbett & Anderson, 1995) as described in Chapter 6.

Hyper parameters

Except the methods which automatically learn the α parameters by using gradient descent, hyper parameter search is used to determine the best hyper parameters (in term RMSE) for the other methods. For examples, to determine the α, β, δ, γ values, we first start a coarse search from 0.1 to 0.9 with 0.1 increment-step. After finding

the best values, we again do a fine-grained search with step-length 0.01 around those values. The history length L is searched in the set {20, 40, 60, 80, 100, 150, 200 }. Since the target variable is the student performance $p \in [0..1]$, the forecasting value is bounded with 0/1 in case it exceeds the interval [0..1].

For the BKT-BF and BKT-EM, we use the same procedures as described in section 5.5 of Chapter 5.

7.4.2 Experimental Results

Figure 7.4 presents the typical RMSE of forecasting using all historical data controlled by the history length L (denoted as *histLength*) and forecasting using all historical data in the same unit and section (denoted as *secLength*), which we have previously illustrated in Figure 7.3. Clearly, the results of *secLength* are not better than *histLength*. This could implicitly mean that to solve the new problems, the learners need all the previous cumulated knowledge rather than only the knowledge in that unit and section. We use the *histLength* strategy for the rest experiments.

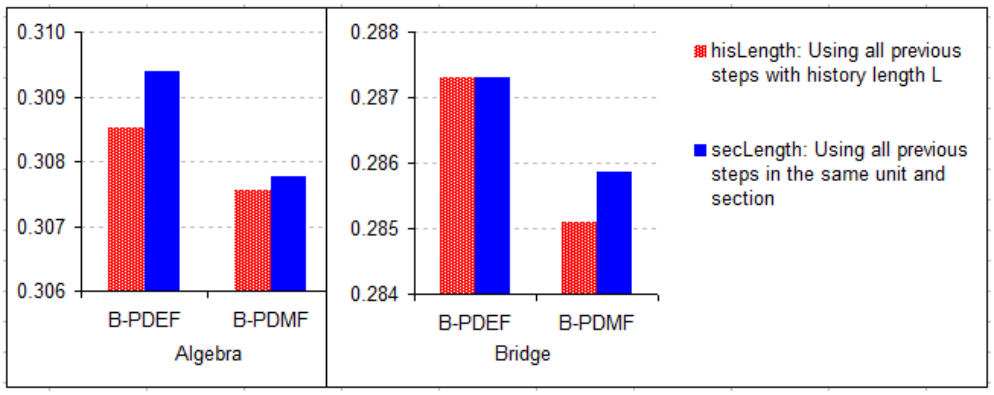


Figure 7.4: RMSE: *histLength* vs. *secLength*

In the experimental datasets, the true target variable is encoded by binary values, i.e., 0 (incorrect) and 1 (correct), thus, the student performance does not show the trend line when visualizing these data sets. To examine how the sequential effect affects to the performance of the learners, we aggregate the performance of all steps in the same problem to a single value and plot the aggregated performance to Figure 7.5. We can clearly see the sequential effect on the sequence of solving problems (from left to right). The average performance shows the trend line, which implicitly means that forecasting methods are appropriate to cope with PSP.

Please note that by aggregating, we will come up with the new data sets and the task now is to predict/forecast the whole problem instead of predicting/forecasting the single step in that problem. This work, however, is out of the scope of this chapter.

Table 7.2 compares the root mean squared error (RMSE) of original (non-personalized)

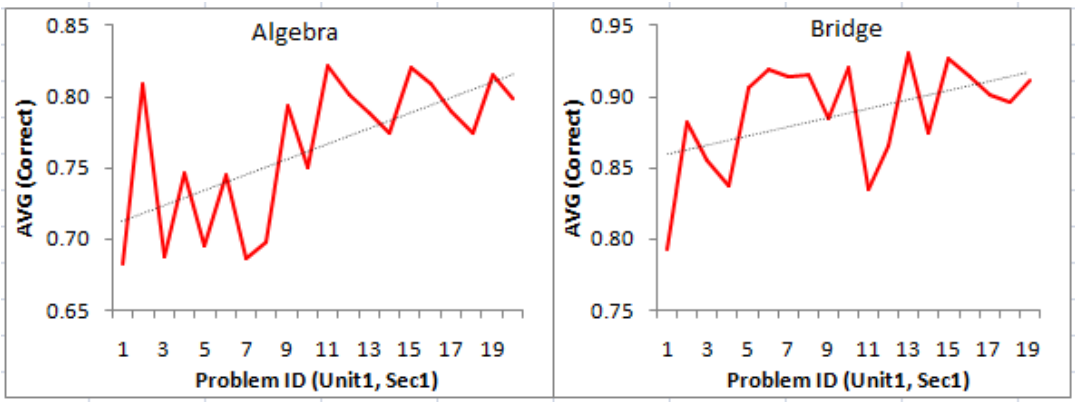


Figure 7.5: Sequential effect on the student performance: $y-axis$ is the average of correct performance and $x-axis$ is the sequence of problems (ID) aggregated from the steps. Typical results of Unit 1 and Section 1

Table 7.2: RMSE: Personalized Forecasting (N-PSEF, B-PSEF) vs. Classical (non-personalized) Forecasting (SEF, DEF)

Approach	Method	Algebra	Bridge	ASSISTments
Non-personalized Forecasting	SEF	0.33439	0.31012	0.47766
	DEF	0.32953	0.30472	0.45954
Personalized Forecasting	N-PSEF	0.33022	0.30433	0.46775
	N-PSEF-SGD	0.33013	0.30420	0.46618
Biased Personalized Forecasting	B-PSEF	0.30767	0.28538	0.41325
	B-PSEF-SGD	0.30746	0.28570	0.41076
	B-PDEF	0.30849	0.28883	0.41490
	B-PDMF	0.30756	0.28510	0.41138

forecasting (SEF and DEF) with personalized forecasting methods (N-PSEF and B-PSEF). The non-personalized double forecasting DEF, which takes the trend into account, has a small improvement comparing to the single SEF, and the personalized methods outperform the non-personalized ones. Moreover, using personalized forecasting and taking into account the "student and task effects" by employing the biased terms, the results (e.g. B-PSEF) significantly improve compared to the non-personalized SEF and DEF.

Figure 7.6 compares the RMSE of personalized forecasting (B-PSEF, B-PDEF, B-PDMF) with the other methods. The proposed methods also outperform the others including the state-of-the-art matrix factorization.

Now, let us compare the personalized forecasting methods with the state-of-the-art

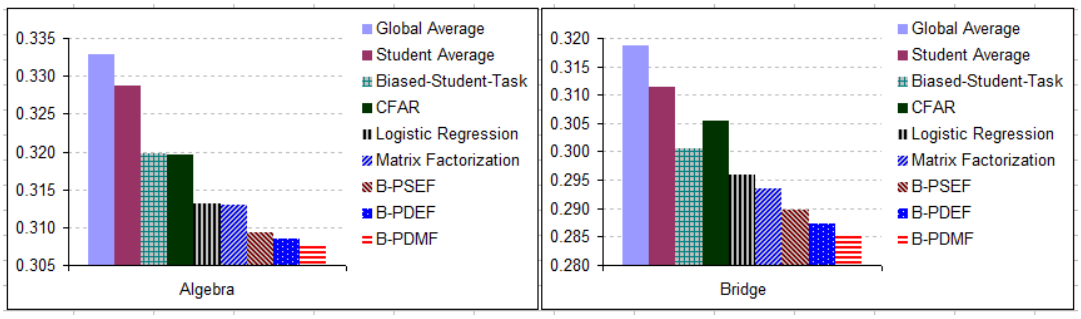


Figure 7.6: RMSE: Personalized Forecasting vs. other methods

BKT-BF (Baker *et al.*, 2008b; Corbett & Anderson, 1995) and the BKT-EM (Chang *et al.*, 2006; Corbett & Anderson, 1995), in Figure 7.7.

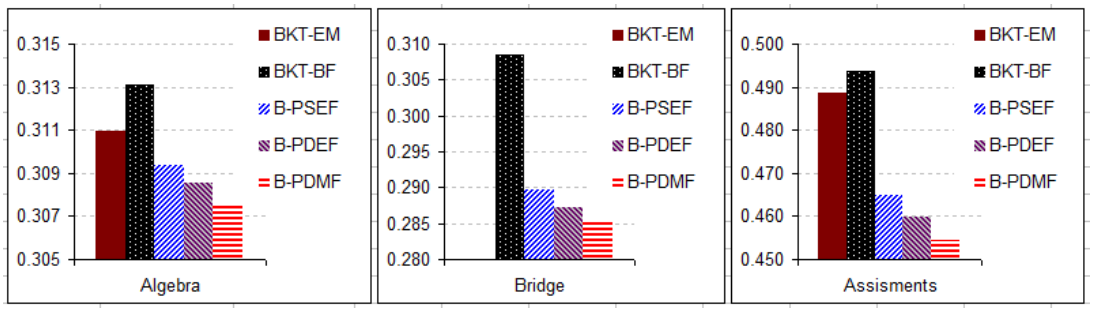


Figure 7.7: RMSE: Personalized Forecasting vs. Bayesian Knowledge Tracing

From the results we can also see that the proposed methods have improvements compared to the state-of-the-art Bayesian Knowledge Tracing models.

7.5 Conclusions

In this chapter, we have introduced the personalized forecasting techniques which are very simple and very fast but quite efficient, for predicting/forecasting student performance. These methods not only take into account the sequential/temporal effect but also take into account the student and task effects.

Experimental results on large data sets show that the personalized forecasting methods perform nicely and much faster than the other state-of-the-art methods in predicting student performance. Thus, these methods would be promising for predicting student performance, especially when the cumulative knowledge of the students (learners) should be taken into account.

So far, we have introduced using the latent factor models for student modeling in Chapters 5 and 6. In this chapter, we have presented the personalized forecasting

techniques to model the sequential/temporal effect. An open issue would be how to take into account both the (student/task) latent factors and the temporal/sequential effect, e.g., by incorporating the forecasting approach into the latent factor models. We will present the solutions for this issue in the next chapter.

Chapter 8

Temporal Effects in Student Modeling

Contents

8.1 Problem Formulation Extension 84
8.2 Canonical Tensor Factorization Models 85
8.3 Tensor Factorization - Averaging Approach (TFA) 86
8.4 Tensor Factorization - Weighting Approach (TFW) 87
8.5 Tensor Factorization - Moving Average Forecasting (TF-MAF) 87
8.6 Tensor Factorization Forecasting (TFF) 88
8.7 Experiments 90
8.7.1 Experimental Setting 90
8.7.2 Experimental Results 91
8.8 Conclusion 93

So far, in Chapter 7 we have presented the *personalized forecasting* techniques to take into account the temporal/sequential effects in predicting student performance, while in Chapter 5 we have introduced the *latent factor models* to implicitly encode the student's and task's latent factors as well as "student effect" and "task effect".

In this chapter, we propose incorporating the forecasting techniques into the latent factor models. In addition, recall that there are two crucial aspects should be taken into account in predicting student performance:

1. The probability of a student to guess correctly while not knowing how to solve the problem at hand or not having the required skills related to the problem ("guess" factor); and the probability of a student to fail while knowing how to solve the

problem or having all of the required skills related to the problem ("slip" factor) (Corbett & Anderson, 1995);

2. The increase in knowledge over time obviously has an effect on a student's performance, e.g., the second time a student does his/her exercises, his/her performance gets better on average. Moreover, from the education point of view, "the more the learners study, the better the performance they get". Thus, the temporal/sequential effect is an important factor to predict the student performance.

Tensor factorization (Kolda & Bader, 2009; Rendle *et al.*, 2009; Skillicorn, 2007), which also belongs to the family of latent factor models, would be a promising approach to deal with these two crucial aspects since they i) are able to implicitly take into account the student/task latent factors, and ii) can incorporate the temporal/sequential aspect into their models. Thus, the methods in this chapter will integrate our previous methods together. This mean that, instead of using only two-mode tensor[1] as in Chapter 5, we now add one more mode to the model - the *time* mode, thus, the approach in this chapter is also a general form of matrix factorization.

8.1 Problem Formulation Extension

In Chapter 2, we have formulated the predicting student performance problem in term of two dimensional recommender system problem (2D-RS). In this chapter, we will introduce the techniques which are used in the context of three dimensional recommender system problem (3D-RS, see an example in Figure 8.1). Thus, the problem formulation need to be extended.

Indeed, when comparing to previous methods, a difference here is the additional mode of the tensor, which represents the *time*. Specifically, the time can be exploited by two different ways:

1. **Concrete time**, which represents specific points of time, e.g., representing by the trial/attempt count, or *problem view* or *opportunity count* which are already described in Chapter 3. This kind of time is usually used in context-aware recommender systems, e.g., weekend, weekday, Christmas day, etc (Adomavicius & Tuzhilin, 2011; Gantner *et al.*, 2010b).

2. **Relative time**, which describes sequence (order) of the data, e.g., the sequence of solving problem (or step) in our work. This kind of time is usually used in forecasting techniques as presented Chapter 7, or in modeling sequential data (Bengio, 1999; Dietterich, 2002; Rendle *et al.*, 2010).

For the **concrete time**, besides the set of students (S), set of tasks (I), and set of performances (P) as described in Chapter 2, we additionally use T as a set of times.

[1]Note: Tensor is also known as a cube; mode is also called dimension; and thus, two-mode tensor is a matrix; in this work, we only use three-mode tensor

Let

$$\mathcal{D}^{train} \subseteq (S \times I \times T \times P)$$

be the observed student performances and

$$\mathcal{D}^{test} \subseteq (S \times I \times T \times P)$$

be the unobserved student performances.

Then the problem of predicting student performance is, given $\mathcal{D}^{train}$ to find

$$\hat{p} : S \times I \times T \to \mathbb{R}$$

such that the measure $\mathcal{E}(\hat{p}, p)$ is satisfied a certain condition, e.g. $\mathcal{E}$ needs to be minimum for the RMSE or MAE, as in equations 2.1 and 2.2, respectively; where $\hat{p}$ is a predicted performance and p is the true performance determined on $\mathcal{D}^{test}$

$$p : S \times I \times T \to P, \qquad (s, i, t) \mapsto p$$

For the **relative time**, the only difference from the formulation in Chapter 2 is to represent the **sequence** of the data. Thus, we only need to change the formulation of the train set and the test set, which are now denoted as

$$\mathcal{D}^{train} \subseteq (S \times I \times P)^*$$

and

$$\mathcal{D}^{test} \subseteq (S \times I \times P)^*$$

In this chapter, we focus on the **relative time**. The use of the **concrete time** will be discussed in Chapter 9.

8.2 Canonical Tensor Factorization Models

Given a three-mode tensor $\mathcal{Z}$ of size $|S| \times |I| \times |T|$, where the first and the second mode describe the student and the task as in previous chapters, respectively; and the third mode describes the concrete time.

Then $\mathcal{Z}$ can be written as a sum of rank-1 tensors by using the Tucker tensor (Tucker, 1966) or by using the CANDECOM-PARAFAC tensor (Carroll & Chang, 1970; Harshman, 1970):

$$\mathcal{Z} \approx \sum_{k=1}^{K} \lambda_k \; w_k \circ h_k \circ q_k \tag{8.1}$$

where λ_k is a scalar vector; $\circ$ is an outer product; and each vector $w_k \in \mathbb{R}^{|S|}$, $h_k \in \mathbb{R}^{|I|}$, and $q_k \in \mathbb{R}^{|T|}$ describes the latent factors of student, task, and time, respectively. An illustration of tensor decomposition is presented in Figure 8.1.

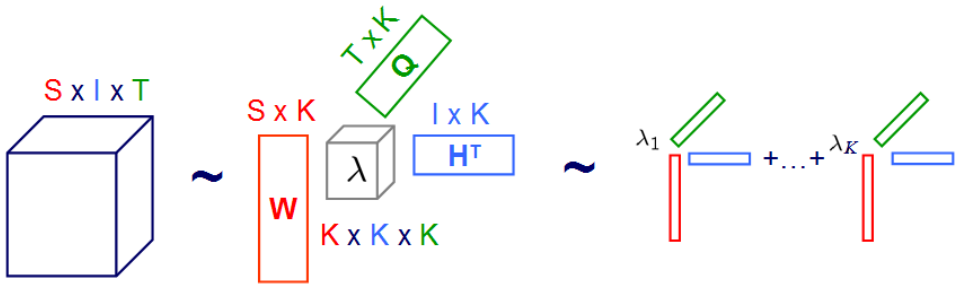


Figure 8.1: CANDECOM-PARAFAC method: A tensor is decomposed into three low-rank matrices.

The tensor factorization (decomposition) approach has been used in many other areas such as recommender systems, topic modeling, and more (Cichocki *et al.*, 2009; Dunlavy *et al.*, 2011; Karatzoglou *et al.*, 2010; Kolda & Bader, 2009; Rendle & Schmidt-Thieme, 2010). However, most of them are used for aforementioned concrete time.

We adopt the idea from this tensor factorization approach to model the temporal/sequential effect in predicting student performance by using the relative time.

8.3 Tensor Factorization - Averaging Approach (TFA)

The simple scoring method is to average on the time mode. This approach was used in Dunlavy *et al.* (2011) for link prediction in term of concrete time[1]. Here, we use it for modeling the relative time.

$$\hat{p}_{siT} = \sum_{k=1}^{K} w_{sk} h_{ik} \Phi_{Tk} \tag{8.2}$$

where w_s and h_i are the student latent factor and the task latent factor as already used in Chapter 5; K is the number of latent factors; T is the time in a sequence which we want to predict; and Φ_{Tk} is determined by

$$\Phi_{Tk} = \frac{\sum_{t=1}^{L} q_{(T-t)k}}{L} \tag{8.3}$$

where q_t is a latent factor vector representing the time, and L is the history length as already described in Section 7.2.1 of Chapter 7, e.g., the number of solving-steps that we want to go back to the history from the current time.

[1]Dunlavy *et al.* (2011) have used a three-mode tensor to represent (author, conference, year), then predicting which authors will publish the papers in which conferences in the future (next year)

We also employ the student bias/effect and the task bias/effect, as described in Section 5.3 of Chapter 5. To take into account these effects, the prediction function for student s on given task i at time T now becomes:

$$\hat{p}_{siT} = \mu + b_s + b_i + \sum_{k=1}^{K} w_{sk} h_{ik} \Phi_{Tk} \tag{8.4}$$

where μ, b_s, and b_i are the global average, student bias, and task bias, respectively. We call this approach **TFA** (Tensor Factorization - Averaging).

8.4 Tensor Factorization - Weighting Approach (TFW)

As mentioned in Section 7.2.4 of Chapter 7, an important factor is that "human memory is limited", so the students may forget what they have studied in the past, e.g., they might perform better on lessons they have learned recently than on such they have learned last year or before. Thus, the recent data may be more important than the old data.

To cope with this, we again use a decay function which reduces the weight θ for the old data. The Φ_{Tk} in equation (8.3) now becomes

$$\Phi_{Tk} = \frac{\sum_{t=1}^{L} q_{(T-t)k} \cdot e^{-t \cdot \theta}}{L} \tag{8.5}$$

For making prediction, the equation 8.4 is still used. We call this approach **TFW** (Tensor Factorization - Weighting)

8.5 Tensor Factorization - Moving Average Forecasting (TFMAF)

We now introduce the model that incorporates the forecasting technique into the latent factor model. Instead of weighting (as the TFW) or averaging (as the TFA) on the time mode, which is presented in the previous sections, we use the forecasting techniques.

However, for simplification purpose, we have just used the Moving Average forecasting[1] with a period L on the time mode. The other forecasting techniques could also be applied in the similar way.

Using moving average, the Φ_{Tk} in equation (8.3) now becomes

$$\Phi_{Tk} = \frac{\sum_{t=1}^{L} q_{tk} \cdot p^{s}_{T-t}}{L} \tag{8.6}$$

[1]Moving average is an unweighted mean of previous n data points in the sequence. Please refer back to the Section 7.1 for details

where T is the time in the sequence at which we want to predict; q_{tk} is the time latent factor; p^s_{T-t} is the performance of student s at the previous time in a sequence; and L is the history length, as used in the previous sections.

For making prediction by taking into account the student effect and the task effect, the equation 8.4 is still used. We call this method **TFMAF** (Tensor Factorization - Moving Average Forecasting).

8.6 Tensor Factorization Forecasting (TFF)

In recommender systems (e-commerce area), Rendle *et al.* (2010) have used matrix factorization with Markov chains to model sequential behavior by learning a transition graph over items that is used to predict the next action based on the recent actions of a user. The authors proposed using previous "basket of items" to predict the next "basket of items" with high probabilities that the users might want to buy.

We adopt this idea, however, in educational environment, one natural fact is that the performance of the students not only depends on the recent knowledge (e.g. the knowledge in the previous problems or sections or units, etc, which act as "previous basket of items") but also depends on all the cumulative knowledge in the past that the students have studied. Thus, we need to adapt this method by using all previous performances of a student which are controlled by the history length L to forecast his/her next performance.

With this approach, the Φ_{Tk} in equation (8.6) now becomes:

$$\Phi_{Tk} = \frac{\sum_{t=1}^{L} h'_{(T-t)k} \cdot q_{tk} \cdot p^s_{T-t}}{L} \tag{8.7}$$

where $h'_{(T-t)k}$ is the latent factor of the previous solved tasks in the sequence.

It is important to note that, here, we use the h' which represents for the previous task latent factors instead of using w' which represents for the previous student latent factors since we thought that, to solve the current task, the student may need his/her own accumulative knowledge from the previous tasks in the sequence (rather than the knowledge from the previous students!). We call this method **TFF** (Tensor Factorization Forecasting).

Similar to the MF and BMF in Chapter 5, we use the stochastic gradient descent to optimize the model parameters (in term of RMSE). We describe the learning algorithm for the TFF method in the following, however, the remaining methods (TFA, TFW, TFMAF) can be done in a similar way.

The TFF's error function which we would like to optimize is presented as

$$\mathcal{O}^{TFF} = \sum_{(s,i,p)\in\mathcal{D}^{train}} e^2_{siT} + \lambda \left(||\mathbf{W}||^2_F + ||\mathbf{H}||^2_F + ||\mathbf{H'}||^2_F + ||\mathbf{Q}||^2_F + b_s^2 + b_i^2\right)$$

Algorithm 7 Learn a tensor factorization forecasting using stochastic gradient descent with K latent factors, β learning rate, λ regularization weight, L history length, and stopping condition

procedure TFF($\mathcal{D}^{train}$, K, β, λ, L, stopping condition)
  $\{\mathbf{W}, \mathbf{H}, \mathbf{H'}, \mathbf{Q}\} \leftarrow \mathcal{N}(0, \sigma^2)$
  $\mu \leftarrow \frac{\sum_{p \in \mathcal{D}^{train}} p}{|\mathcal{D}^{train}|}$
  for each student s **do**
    $b_s \leftarrow \frac{\sum_i (p_{si} - \mu)}{|\mathcal{D}_s^{train}|}$
  end for
  for each task i **do**
    $b_i \leftarrow \frac{\sum_s (p_{si} - \mu)}{|\mathcal{D}_i^{train}|}$
  end for
  while (stopping condition is NOT met) **do**
    Draw randomly (s, i, p_{siT}) at row T from $\mathcal{D}^{train}$
    ▷ *T is considered as current time in the sequence*
    $$\hat{p}_{siT} \leftarrow \mu + b_s + b_i + \sum_{k=1}^{K} \left(w_{sk} h_{ik} \left(\frac{\sum_{t=1}^{L} h'_{(T-t)k} \cdot q_{tk} \cdot p^s_{T-t}}{L} \right) \right)$$
    $e_{siT} \leftarrow p_{siT} - \hat{p}_{siT}$
    $\mu \leftarrow \mu + \beta \cdot e_{siT}$
    $b_s \leftarrow b_s + \beta \cdot (e_{siT} - \lambda \cdot b_s)$
    $b_i \leftarrow b_i + \beta \cdot (e_{siT} - \lambda \cdot b_i)$
    for $k \leftarrow 1, \ldots, K$ **do**
      $w_{sk} \leftarrow w_{sk} - \beta \left(\frac{\partial \mathcal{O}^{\text{TFF}}}{\partial w_{sk}} \right)$
      $h_{ik} \leftarrow h_{ik} - \beta \left(\frac{\partial \mathcal{O}^{\text{TFF}}}{\partial h_{ik}} \right)$
      for $t \leftarrow 1, \ldots, L$ **do**
        $h'_{(T-t)k} \leftarrow h'_{(T-t)k} - \beta \left(\frac{\partial \mathcal{O}^{\text{TFF}}}{\partial h'_{(T-t)k}} \right)$
        $q_{tk} \leftarrow q_{tk} - \beta \left(\frac{\partial \mathcal{O}^{\text{TFF}}}{\partial q_{tk}} \right)$
      end for
    end for
  end while
  return $\{\mathbf{W}, \mathbf{H}, \mathbf{H}', \mathbf{Q}, b_s, b_i, \mu\}$
end procedure

where the second term λ ($||\mathbf{W}||_F^2 + ||\mathbf{H}||_F^2 + ||\mathbf{H}'||_F^2 + ||\mathbf{Q}||_F^2 + b_s^2 + b_i^2$) is the regularization which is used to prevent over-fitting, and the first term e_{siT}^2 is the squared

error, which is determined by

$$e_{siT}^2 = (p_{siT} - \hat{p}_{siT})^2 = (p_{siT} - \mu - b_s - b_i - \sum_{k=1}^{K} w_{sk} h_{ik} \Phi_{Tk})^2$$

The TFF iteratively updates its latent factors using the following gradients

$$\frac{\partial \mathcal{O}^{TFF}}{\partial w_{sk}} = -2 e_{siT} h_{ik} \Phi_{Tk} + \lambda w_{sk} \tag{8.8}$$

$$\frac{\partial \mathcal{O}^{TFF}}{\partial h_{ik}} = -2 e_{siT} w_{sk} \Phi_{Tk} + \lambda h_{ik} \tag{8.9}$$

$$\frac{\partial \mathcal{O}^{TFF}}{\partial h'_{(T-t)k}} = -2 e_{siT} w_{sk} h_{ik} \left(\frac{\sum_{t=1}^{L} q_{tk} \cdot p_{T-t}^{s}}{L} \right) + \lambda h'_{(T-t)k} \tag{8.10}$$

$$\frac{\partial \mathcal{O}^{TFF}}{\partial q_{tk}} = -2 e_{siT} w_{sk} h_{ik} \left(\frac{\sum_{t=1}^{L} h'_{(T-t)k} \cdot p_{T-t}^{s}}{L} \right) + \lambda q_{tk} \tag{8.11}$$

Algorithm 7 briefly summarizes the training process of the TFF. First, the parameters (**W,H,H',Q**) are initialized randomly from the normal distribution $\mathcal{N}(0, \sigma^2)$ with mean is 0 and standard deviation $\sigma^2 = 0.01$, as in line 2. Then, we compute the values of the global average, student bias, and task bias as in lines 3-9. While the stopping condition is not met, e.g., not reaching the predefined number of iterations or not converging ($\mathcal{O}^{\text{TFF}}_{Iteration_{(n-1)}} - \mathcal{O}^{\text{TFF}}_{Iteration_n} < \epsilon$), the latent factors and the biased terms are updated iteratively.

After the training phase, the parameters are obtained. Then, we can easily predict the student performance using equation 8.4.

8.7 Experiments

In this section, we first present how to set up the experiments, we then provide several experimental results to validate the proposed approach.

8.7.1 Experimental Setting

We still use the same setting as described in Section 5.5.2 of Chapter 5.

Moreover, in the specific data sets used for experiments, the actual target variable (the actual performance - *correct first attempt*) is encoded by 0 (incorrect) and 1 (correct), so we modify the equations (8.6) and (8.7) to avoid the zero value in the factor product. The Φ_{Tk} in equation (8.6) now becomes:

$$\Phi_{Tk} = \frac{\sum_{t=1}^{L} q_{tk} \cdot (p_{T-t}^{s} - 0.5) \cdot 2}{L}$$

and the Φ_{Tk} in equation (8.7) now becomes:

$$\Phi_{Tk} = \frac{\sum_{t=1}^{L} h'_{(T-t)k} \cdot q_{tk} \cdot (p_{T-t}^{s} - 0.5) \cdot 2}{L}$$

However, this is just a simple heuristic, other modifications on these specific data sets could also be used.

8.7.2 Experimental Results

Besides the finding results about temporal effect on student performance as in Chapter 7, we now analyze the temporal effect in another way.

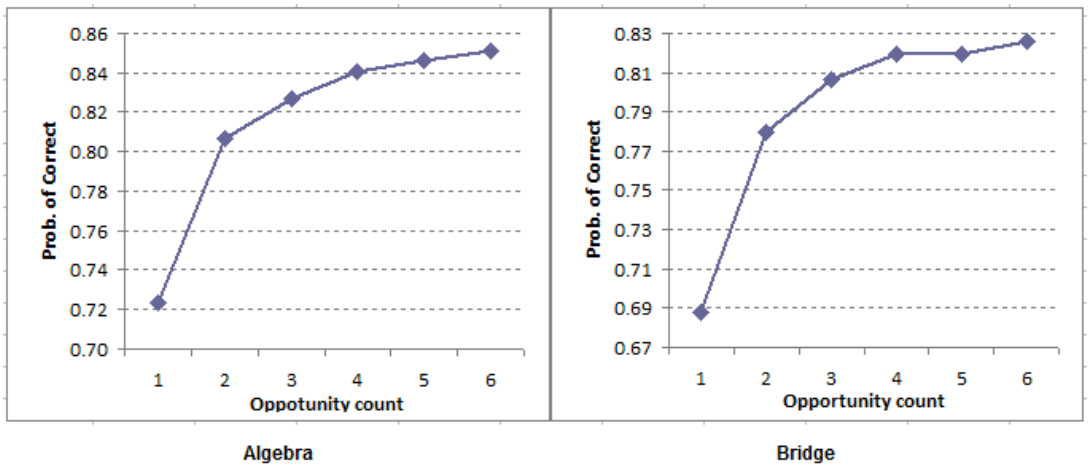


Figure 8.2: Temporal effect on student performance

Figure 8.2 describes the effect of the time on knowledge of the students for both Algebra and Bridge data sets. In this figure, the *x-axis* is the number of times that the students have chances to learn the skills (the "*opportunity count*" column in these data sets), the *y-axis* is the ratio of the number of students solving the problem correctly. Clearly, we can see that their performance has been improved when they have more opportunities to learn the skills. This trend may reflect the educational factor that "the more the students learn, the better the performance they get".

Figure 8.3 presents the root mean squared error (RMSE) of the proposed tensor factorization methods which factorize on the student (as *user*), solving-step I_7 (as *item*), and the sequence of solving-step (as *time*) for both Algebra and Bridge data sets. The results of the proposed methods are improved compared to the others.

Moreover, when comparing to the matrix factorization which does not take into account the temporal effect, the tensor factorization methods have also been improved. These results may implicitly reflect the natural fact that we have mentioned before: "the knowledge of the student improves over time". However, the results of the four tensor factorization methods (TFA, TFW, TFMAF, and TFF) are not far from each other on Algebra data set.

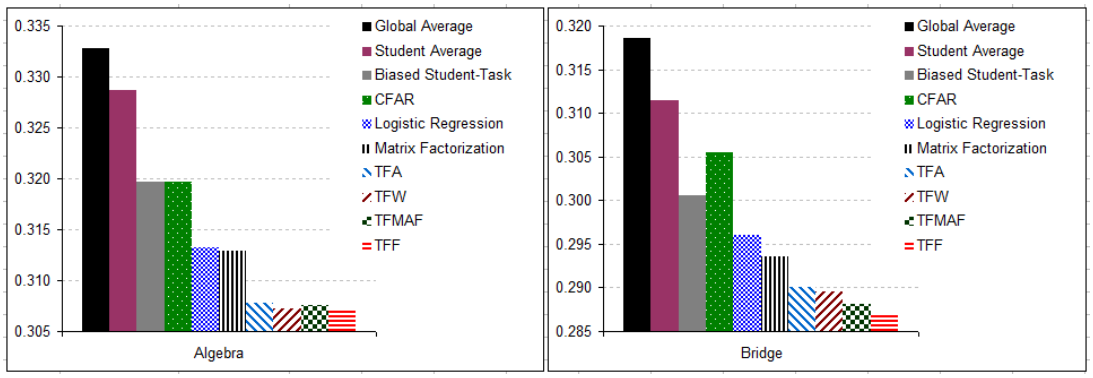


Figure 8.3: RMSE: Modeling temporal effect using tensor factorization

Table 8.1: RMSE: Bayesian Knowledge Tracing vs. Tensor Factorization Models

Data set	BKT-EM	BKT-BF	TFA	TFW	TFMAF	TFF
Algebra	0.31098	0.31308	0.30777	0.30724	0.30755	**0.30697**
Bridge	N/A	0.30849	0.29000	0.28960	0.28808	**0.28702**
ASSISTments	0.48860	0.49353	0.45671	0.45695	0.45568	**0.45566**

Now, let us comparing the proposed tensor factorization methods with the well-known Bayesian Knowledge Tracing: BKT-EM (Chang *et al.*, 2006) and BKT-BF (Baker *et al.*, 2008b) in Table 8.1. Again, we have not reported the BKT-EM on Bridge data set since it was intractable. From the empirical results we can also see that the tensor factorization models have improvements compared to the Bayesian Knowledge Tracing models, and the TFF outperforms the other ones on these data sets.

For referencing, we report the hyper parameters which we used in Table 8.2. The running time approximation of tensor factorization methods are high, e.g. the TFF $\approx$ 15 hours on the largest data set - Bridge). However, in educational environment where the models need not to be retrained continuously as in e-commerce area, this running time should not be an issue.

Table 8.2: Hyper parameters of the tensor factorization methods

Method	Data set	Hyper parameters
TFA	Algebra	β=0.001, #iter=20, K=32, λ=0.001, L=8
TFW	Algebra	β=0.001, #iter=30, K=16, λ=0.015, L=8, θ= 0.2
TFMAF	Algebra	β=0.015, #iter=30, K=16, λ=0.015, L=8
TFF	Algebra	β=0.001, #iter=60, K=16, λ=0.015, L=10
TFA	Bridge	β=0.01, #iter=30, K=32, λ=0.015, L=8
TFW	Bridge	β=0.01, #iter=30, K=32, λ=0.015, L=8, θ= 0.4
TFMAF	Bridge	β=0.005, #iter=20, K=64, λ=0.015, L=10
TFF	Bridge	β=0.0015, #iter=60, K=16, λ=0.005, L=5
TFA	ASSISTments	β=0.01, #iter=50, K=16, λ=0.015, L=24
TFW	ASSISTments	β=0.01, #iter=20, K=8, λ=0.015, L=16, θ= 0.2
TFMAF	ASSISTments	β=0.01, #iter=30, K=16, λ=0.015, L=2
TFF	ASSISTments	β=0.01, #iter=50, K=16, λ=0.015, L=24

β: learning rate, λ: regularization; K: the number of factors;
$\#iter$: the number of iterations; L: history length; θ: weight value

8.8 Conclusion

From educational point of view, the learner's knowledge improves and cumulates over time, thus, sequential effect is an important information for predicting student performance. In this chapter, we have proposed a new approach which use tensor factorization to take into account the sequential/temporal effect.

In this work, a simple forecasting technique, which is moving average, was incorporated into the factorization model. However, applying more sophisticated forecasting techniques, e.g. the Holt-Winter (Chatfield & Yar, 1988) as in Dunlavy *et al.* (2011), may produce better results. Moreover, taking into account both the multi-relational aspect as in Chapter 6 and the temporal effect as in this chapter may also get further improvements.

Chapter 9

Context-Aware Factorization Models for Student's Task Recommendation

Contents

9.1 Introduction 94
9.2 Related Work 96
9.3 Problem Reformulation 96
9.4 None-Context Approach (Baseline) 97
9.5 Context-Aware Factorization Models 97
9.5.1 Pre-Filtering 98
9.5.2 Contextual Modeling 98
9.6 Experiments 98
9.6.1 Data Sets 98
9.6.2 Experimental Setting 99
9.6.3 Experimental Results 99
9.7 Discussion 100

9.1 Introduction

In Chapter 5, we have introduced using collaborative filtering (User/Item-kNN and latent factor models) for predicting student performance. These techniques, obviously, can also be used for recommending the tasks to the students.

In principle, *collaborative filtering makes an assumption that each user rates for an item once*, which means that it only takes into account the last rating (i.e. the most recent rating) of the users and ignores all the previous ratings. However, in educational

environment, for example, recommending the tasks (or problems, exercises, etc) to the students, this assumption may not hold since each student may perform a specific task several times, and thus, there are *multiple interactions* between the student and the task.

As already discussed in Chapters 7 and 8, there are two important issues in predicting student performance: *i)* the student performance (student knowledge) improves and cumulates over time and *ii)* the second time the student is doing his/her exercises, his/her performance gets better on average. Therefore, we should take into account not only the *sequential/temporal effects* but also the *multiple interactions* between the students and the tasks. Without utilizing these interactions by directly applying the traditional collaborative filtering techniques for predicting student performance may produce unsatisfied results.

In this chapter, we propose using *context-aware factorization models to utilize the multiple interactions (performances)* of the given student-task pairs. The proposed approach can be used not only for predicting student performance but also for personalized learning environments (e.g., recommending the tasks to the students).

Furthermore, this chapter also opens an issue for recommendation in e-learning, that is, *task recommendation based on student performance*. Indeed, as mentioned in the literature, recommender systems for educational purposes are more complex and challenging research direction compared to the case of e-commerce in which recommender systems are widely being used (Drachsler *et al.*, 2009). The reasons are that the learners have specific learning goals that they want to achieve within a specified competence in a certain time, and that the preferred learning activities of learners might not be the pedagogically the most adequate (Tang & McCalla, 2004). Thus, the recommendations for technology enhanced learning scenarios have differences from those in other domains because recommendations in e-learning should be guided by educational objectives, and not by the user's preferences (Santos & Boticario, 2010).

Our approach which uses the student performance for "task recommendation" would be a promising approach to tackle the above problems since we can recommend the tasks to the students based on their *performances* instead of their *preferences*. With this approach, one can recommend the similar tasks to the students as well as determine which tasks are notoriously difficult for a given student. For example, there is a large bank of exercises where the students lose lots of time to solve the problems which are too easy or too hard for them. When a system is able to predict the student performance, it could recommend more appropriate exercises to the students, e.g., by filtering out the tasks with predicted high performance (since these tasks are too easy) or filtering out the tasks with predicted low performance (since they are too hard) or both, depending on the goals of the e-learning system.

However, in this work, we have not focused on building a real recommender system, but on modeling the student's task recommendation using context-aware factorization approach.

9.2 Related Work

Recommender systems have been applied to technology enhanced learning, especially e-learning, recently (Manouselis *et al.*, 2010). The main goal of recommender systems for e-learning is to recommend materials / resources[1] (e.g. papers, books, hyper-links,...) (Ghauth & Abdullah, 2010; Luo *et al.*, 2010; Tang & McCalla, 2005; Zaiane, 2002), course enrollment (Garcia *et al.*, 2009; OMahony & Smyth, 2007), and more (Ghauth & Abdullah, 2010; Manouselis *et al.*, 2010) to the learners in both formal and informal learning environment (Drachsler *et al.*, 2009).

More precisely, Garcia *et al.* (2009) use association rule mining to discover interesting information through student performance data in the form of IF-THEN rules, followed by generating recommendations based on those rules; Bobadilla *et al.* (2009) proposed an equation for collaborative filtering which incorporated the test score from the learners into the item prediction function; Ge *et al.* (2006) combined the content-based filtering and collaborative filtering to personalize the recommendations for a courseware selection module; Soonthornphisaj *et al.* (2006) applied collaborative filtering to predict the most suitable documents for the learners; while Khribi *et al.* (2008) employed web mining techniques with content-based and collaborative filtering to compute the relevant links for recommending to the learners. This use of recommender systems usually corresponds to the so-called item recommendation (or item prediction) task in the recommender literature.

In another work, Cetintas *et al.* (2010) proposed a temporal collaborative filtering approach to automatically predict the correctness of students' problem solving in an intelligent math tutoring system. This approach utilized the multiple interactions for a student-problem pair, however, the authors used the k-nearest neighbors collaborative filtering (kNN-CF).

As seen, most of the literature have used the kNN-CF, which we have pointed out its drawback in the previous section, for generating the recommendations. Moreover, what so far is missing in e-learning systems is the exploitation of the recommender system capability to predict the scores - the task called rating prediction.

In this chapter we propose to improve the rating prediction by using the context-aware factorization models, however, the other context-aware methods can be found in Adomavicius & Tuzhilin (2011).

9.3 Problem Reformulation

In this work, the proposed context-aware models also use the tensor factorization approach. Thus, we use the same formulation for modeling the *concrete time* on the third mode of the tensor as described in Section 8.1 of Chapter 8. The tensor is represented by: student s, problem i, problem view t which tracks how many times the student has interacted with the problem, and the performance p ($p \in [0..1]$).

[1] The literature call them Learning Objects (LO)

Without using context, these data can be mapped as: student s becomes the user and problem i becomes item, which are presented in a matrix of (s, i, p) as in Figure 9.1a.

For utilizing the *context* t (problem view in this case) which represents the *multiple interactions* between the student and the task, these data are presented in a three-mode tensor of (s, i, t, p), as illustrated in Figure 9.1c.

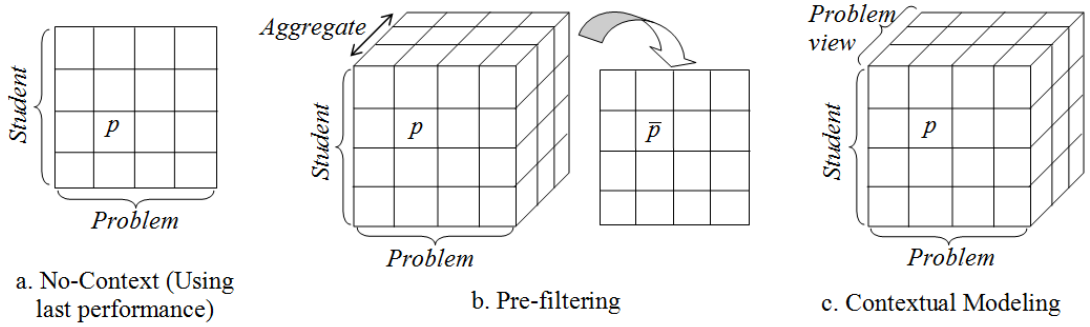


Figure 9.1: An illustration of None-Context vs. Context-aware approach

9.4 None-Context Approach (Baseline)

Traditional collaborative filtering has an assumption that each user rates for an item once, which means that only the last rating (the most recent rating) is used. Similarly, as a baseline in this work, the last performance p of a student-problem pair (s, i) is used (by this way, we ignore the multiple interactions between the students and the problems). Finally, a matrix factorization model as described in Chapter 5 is applied.

As introduced in Chapter 5, using matrix factorization the performance given by a student s to a problem i is predicted by:

$$\hat{p}_{si} = \sum_{k=1}^{K} w_{sk} h_{ik} \tag{9.1}$$

where $\mathbf{W}$ and $\mathbf{H}$ are model parameters which can be obtained by an optimization process using either stochastic gradient descent (Bottou, 2004) or Alternating Least Squares (Kolda & Bader, 2009) given a criterion such as root mean squared error (RMSE) or mean absolute error (MAE).

9.5 Context-Aware Factorization Models

In this chapter, we make use of two context-aware models: "Pre-Filtering" and "Contextual modeling" as named in the literature (Adomavicius & Tuzhilin, 2011). Please

note that we propose using factorization approach instead of heuristic-based and model-based approaches as in Adomavicius & Tuzhilin (2011).

9.5.1 Pre-Filtering

As its name, this method requires pre-processing on the data sets. To do this, the performance p is aggregated (e.g., averaged) along the context t. Thus, the three-mode tensor represented by (s, i, t, p) now becomes a two-mode tensor (a matrix) which is represented by (s, i, $\bar{p}$) as illustrated in Figure 9.1b; where $\bar{p}$ is averaging from p for each student-problem pair.

After the pre-filtering step, we apply the matrix factorization method to factorize on student-problem pairs as already described in Section 9.4.

9.5.2 Contextual Modeling

In this method, the context t which represents the multiple interactions between student s and problem i is preserved. Thus, we now have to deal with a three-mode tensor (three-dimensional recommender systems) as described in Chapter 8. (please note that, here, we use the *concrete time* instead of the *relative time* as in Chapter 8).

As introduced in Chapter 8, the performance of student s for problem i at context t is predicted by:

$$\hat{p}_{sit} = \sum_{k=1}^{K} \lambda_k w_{sk} h_{ik} q_{tk} \tag{9.2}$$

After the prediction phase, we can filter out the tasks with predicted high performance since these tasks are too easy, or filter out the tasks with predicted low performance (too hard) or both, depending on the goals of the e-learning system. Thus, the appropriate tasks can be delivered to students.

9.6 Experiments

To validate the proposed approach, we conduct some experiments as in the following.

9.6.1 Data Sets

For experiments, we also use two data sets from the KDD Cup 2010, the Algebra and Bridge, as described in Chapter 3. However, we now use these data sets with different purpose, that is, predicting for a whole *problem* instead of predicting for a single step in a problem as in all previous chapters. Thus, this task is completely different to the task that we have done before since our purpose now is to predict and recommend the similar *problems* to the students.

For this purpose, we have just selected the first 5,000 problems in both the Algebra and Bridge data sets. Moreover, when representing to a tensor, the third mode - context t (problem view) - is very sparse. One simple method for dealing with this sparsity is aggregating the context to some "bins". For example, for a given student-problem pair, if we have 15 "interactions" (problem views) we can simply aggregate them into 3 bins, e.g. [1-5], [5-10], and [10-15]. Nevertheless, as introduced at the beginning, one of the main goal of this chapter is to state that by utilizing the multiple interactions between the student and the task using the context-aware factorization approach, one can improve the prediction results. Thus, for convenience purpose, we simply select all problems in the data sets which have maximum two interactions (problem views).

9.6.2 Experimental Setting

The baseline for comparison is using the last performance (in this case, the performance on the last problem view) from the data sets described in previous section. Then applying matrix factorization on these data.

We use 3-fold cross-validation and paired t-test with significance level 0.05 for all experiments. We also do hyper parameter search to determine good hyper parameters for all methods. For diversity in the experiments, we now use the Matlab Tensor Toolbox[1] (Kolda & Bader, 2009), which uses Alternating Least Squares instead of Stochastic Gradient Descent as in previous chapters.

9.6.3 Experimental Results

Tables 9.1 and 9.2 present the mean absolute error (MAE) and the root mean squared error (RMSE) of the context-aware factorization methods (Pre-Filtering and Contextual Modeling) which take into account the multiple interactions of student-problem pairs versus the baseline without using context.

Table 9.1: MAE: None-Context vs. Context-Aware Methods

Data set	None-Context	Pre-Filtering	Contextual Modeling
Algebra	0.247± 0.015	0.239±0.016	**0.239**±0.015
Bridge	0.193± 0.033	0.185±0.030	**0.183**±0.030
Average	0.220	0.212	**0.211**

Clearly, from these results we can see that the context-aware methods can improve the prediction results compared to the none-context method (baseline). Thus, this approach can be a reasonable choice for personalized learning environment, especially for recommending the tasks (or problems, exercises, etc) to the students.

Moreover, the improvements in the prediction models may implicitly mean that the system can recommend the "right tasks" to the students, and thus, we can help the

[1] http://csmr.ca.sandia.gov/∼tgkolda/TensorToolbox/

Table 9.2: RMSE: None-Context vs. Context-Aware Methods

Data set	None-Context	Pre-Filtering	Contextual Modeling
Algebra	0.311± 0.023	0.307±0.023	**0.302**±0.022
Bridge	0.232± 0.002	0.232±0.000	0.232±0.000
Average	0.272	0.269	**0.267**

students reducing their time and effort in solving the tasks, e.g. not providing them the problems that they have already knew.

Using these context-aware models, we can generate the performance for a given student-task pair, so the remaining works are just wrapping around with an interface to deliver the recommendations. However, this work is out of the scope of this chapter, and it would be interesting for future work.

9.7 Discussion

So far, we have introduce using context-aware methods to take into account the multiple interactions between the students and the tasks. By doing so, we can improve the prediction results and thus providing better recommendations to the students.

Using our approach, there are many areas of applications for recommendations in learning environments. For instance, in the example of Figure 5.1, we could recommend to the students some similar exercises which are neither too hard nor too easy for them, thus, the system can help the student improving their discipline/knowledge by learning/doing similar exercises. Moreover, from the scenario in Chapter 2, when the students solve a problem, we could also recommend them some similar problems which may help them consolidate and expand their knowledge. Another example is the recommendation of similar grammar structures, vocabularies, or even a similar problem/section when the student is learning or doing exercises in an English course, etc.

Although the proposed methods can improve the prediction results, there are still available rooms for further improvements. For example, we can employ the student bias and the task bias to the models as presented in previous chapters. In addition, exploiting multiple interactions and multiple relationships at the same time may also achieve better results.

Chapter 10

Learning from Imbalanced Data

Contents

10.1 Introduction 102
10.2 Problem Formulation 103
10.3 State-of-the-art Methods and Related Work 103
10.3.1 Data-Level Approach 104
10.3.2 Classifier-Level Approach 106
10.3.3 Combination Approach 107
10.3.4 Evaluation Measures 109
10.4 Compound Cost-Sensitive Learning Methods 112
10.4.1 Combining Sampling Techniques with CSL (S-CSL) 112
10.4.2 Optimized Cost Ratio for CSL (O-CSL) 115
10.5 Thresholding on Bayesian Posterior Probabilities 117
10.5.1 Learning in Bayesian Networks 118
10.5.2 Thresholding on Bayesian Posterior Probabilities 119
10.5.3 Thresholding on Sampling Data 121
10.6 B42 - A New Evaluation Measure 124
10.7 Experiments 125
10.7.1 Data Sets 126
10.7.2 Experimental Setting 128
10.7.3 Experimental Results 129
10.8 Conclusion 146

In this chapter, we will introduce several methods as well as a new evaluation measure for learning from imbalanced data sets.

10.1 Introduction

In Chapters 1 and 3, we have pointed out that predicting student performance can also be considered as binary classification problem, in which our target variable (e.g., the "correct first attempt") has binary values, e.g., 1 indicating correct answer and 0 indicating incorrect answer.

By doing so, we have discovered an issue in student performance data, which may affect to quality of the prediction results, that is, *class imbalance problem* since there are more examples belonging to class 1 (*majority class*) than the examples belonging to class 0 (*minority class*). This phenomenon appears not only in tutoring systems' data sets but also in academic systems' data sets. For example, the class imbalance problem happens when we predict the student GPA (or CGPA) in a semester/term (or in a year, etc) which is estimated by a ranking measure, e.g., A, or B+, or B, etc, (or Excellent, Very Good, Good, etc, depending on the grading systems) since the number of Excellent students and the number of Good students are rather skewed.

Indeed, the class imbalance problem is a phenomenon in which the class distribution[1] is far from the uniform distribution. It appears in many machine learning applications such as fraud detection, network intrusion detection, oil-spill detection, disease diagnosis, and many other areas (Chawla *et al.*, 2004; He & Garcia, 2009). Most classifiers in supervised machine learning are designed to maximize the accuracy of their models. Thus, when learning from imbalanced data, they are usually overwhelmed by the majority class examples. This is the main cause for the performance degradation of such classifiers, and is also considered as one of ten challenging problems in data mining research[2] (Yang & Wu, 2006).

For example, in fraud credit card detection, suppose that a data set has 999 legitimate transactions (majority class) and only 1 fraudulent transaction (minority class – the one we would like to detect). To maximize the accuracy, in this case, the classifiers which are optimized for the accuracy will classify all transactions as belonging to the majority class to get 99.9% accuracy. However, this result has no meaning because the fraudulent transaction (the most important one) is misclassified.

Obviously, to evaluate the classifiers in this case, the accuracy metric becomes useless, and the area under the ROC curve (AUC) is commonly used instead (Bradley, 1997; Hanley & Mcneil, 1982). However, Hand (2009) has shown that the AUC has a deficiency and proposed the "H measure" to overcome this incoherence[3], which uses a symmetric Beta distribution. When learning from imbalanced data, misclassifying a minority class example (e.g., a fraud credit card transaction) is much more serious than misclassifying a majority example. In this chapter, we propose using an *asymmetric* Beta distribution instead of the symmetric one as in the H measure.

Moreover, to alleviate the class imbalance problem, we also introduce several meth-

[1] We consider the problem of binary classification.

[2] http://www.kdnuggets.com/polls/2005/important_data_mining_topics.htm

[3] We will discuss about this problem later

ods which aim at improving the prediction results. In this chapter, our goal is to provide the methods which can be applied in general cases, e.g., in the other applications of machine learning rather than only for predicting student performance. Thus, most of the experiments have been done on the imbalanced data sets which come from the other domains, e.g., network intrusion detection, customer retention rate detection, etc, and a few experiments are conducted on student performance data.

10.2 Problem Formulation

In binary classification problem, class imbalance is a phenomenon in which the class distribution is far from the uniform distribution. It can also be described as the *majority class* outnumbering of the *minority* one by a factor.

More formally, let $\mathcal{D}$ be a data set consisting of n examples $< x_i, y_i > (i = 1 \cdots n)$, where $x_i \in \mathcal{X}$ are input features and y_i is the target variable.

Let $y_{i+} \in \{+,\ +1,\ 1, \textit{Positive}, \textit{Yes}, \textit{True}\}$ denote the *minority class*

Let $y_{i-} \in \{-,\ -1,\ 0, \textit{Negative}, \textit{No}, \textit{False}\}$ denote the *majority class*

Then the class imbalance problem happens when $|y_{i-}| >> |y_{i+}|$.

Depending on application areas, the ">>" can be quantified by a different amount, which usually is the *imbalanced ratio* or the *percentage of the minority class*, where

$$\textit{imbalanced ratio} = \frac{|y_{i-}|}{|y_{i+}|}$$

and

$$\%\textit{minority} = \frac{|y_{i+}| \cdot 100}{n}$$

and dealing with class imbalance problem is to improve the performance of the classifiers in such imbalanced data set.

Please note that the class labels denoted above are usually used in the literature, however, in student performance prediction problem, the majority class is the "correct answer" which usually is denoted as 1, and the minority class is the "incorrect answer" which usually is denoted as 0. Nevertheless, those notations should not be a problem since we just simply swap the labels.

10.3 State-of-the-art Methods and Related Work

To deal with imbalanced data sets, many papers have been published. Generally, the literature can be categorized in three groups: data level, classifier (algorithm) level, and the combination of these two approaches. At data level, the popular task is to modify the class distributions by using (re)sampling techniques, e.g., over-sampling or under-sampling. At classifier level, many techniques have been introduced. Again, they can be categorized into some groups such as internally manipulating the classifiers, one-class learning, ensemble learning, and cost-sensitive learning.

However, due to the large number of methods and their variants as summarized in the literature (Chawla *et al.*, 2004; He & Garcia, 2009), we only present some of the most popular and the state-of-the-art techniques which we have used for comparisons in our study.

10.3.1 Data-Level Approach

In the data-level approach, the most popular techniques are sampling which includes heuristic or non-heuristic oversampling (Chawla *et al.*, 2002; Nickerson *et al.*, 2001), undersampling (Li *et al.*, 2008a; Raskutti & Kowalczyk, 2004) and data cleaning rules such as removing "noise" and "borderline" examples (Hart, 1968; Tomek, 1976; Wilson, 1972).

Random oversampling **ROS** is a non-heuristic method which is used to balance the class distribution by randomly duplicating the minority class examples until reaching a pre-defined percentage (or a number of examples), usually until two classes are balanced.

A well-known oversampling method was introduced by Chawla *et al.* (2002) called Synthetic Minority Over-sampling Technique (**SMOTE**). This is one of the state-of-the-art methods in dealing with class imbalance problem. The SMOTE generates new artificial minority examples by interpolating between the existing minority examples rather than simply duplicating the original examples. This method, at first, finds k nearest neighbors of each minority example (according to authors, $k = 5$); then, it selects a random nearest neighbor; finally, the new synthetic examples are generated along the line segment joining a minority class sample and its nearest neighbor. The SMOTE is presented in Algorithm 8.

In contrast with ROS, random undersampling (**RUS**) is a sampling method which randomly eliminates the majority class examples until reaching the pre-defined percentage (or examples). However, the random undersampling may discard a lot of useful information.

Another well-know method that can be used as undersampling is Tomek's Link (**TLINK**) (Tomek, 1976). TLINK is a method for cleaning the data by removing noise or borderline examples. Given two examples e_i and e_j belonging to different classes and $d(e_i, e_j)$ is the distance between e_i and e_j. A pair (e_i, e_j) is called a TLINK if there is no example e_l such that $d(e_i, e_l) < d(e_i, e_j)$ or $d(e_j, e_l) < d(e_i, e_j)$. If there is a TLINK between two examples, then either one of them is noise or both of them are borderline examples. In this work, we would like to use TLINK as the undersampling method, so only majority examples will be removed.

The other there methods that can also be used for removing "noise" and "borderline examples" are CNN, ENN and OSS which will be described in the following.

The Condensed Nearest Neighbor Rule (**CNN**) (Hart, 1968) is used to find a consistent subset of examples. A subset $\hat{E} \subseteq \mathrm{E}$ is consistent with E if using the 1-nearest neighbor classifier, $\hat{E}$ correctly classifies the examples in E.

Algorithm 8 SMOTE - Synthetic Minority Over-sampling Technique (Chawla *et al.*, 2002)

procedure SMOTE(T, N, k)

Input: Number of minority class samples T; Amount of SMOTE $N\%$; Number of nearest neighbors k
Output: $(N/100) * T$ synthetic minority class samples

(∗ *If N is less than 100%, randomize the minority class samples as only a random percent of them will be SMOTEd.* ∗)
if $N < 100$
  then Randomize the T minority class samples
    $T = (N/100) * T$
    $N = 100$
endif
$N = (int)(N/100)$ (∗ *The amount of SMOTE is assumed to be in integral multiples of 100.* ∗)
k = Number of nearest neighbors
$numattrs$ = Number of attributes
$Sample[\,][\,]$: array for original minority class samples
$newindex$: keeps a count of number of synthetic samples generated, initialized to 0
$Synthetic[\,][\,]$: array for synthetic samples
(∗ *Compute k nearest neighbors for each minority class sample only.* ∗)
for $i \leftarrow 1$ **to** T
    Compute k nearest neighbors for i, and save the indices in the $nnarray$
    Populate(N, i, $nnarray$)
endfor

Populate(N, i, $nnarray$) (∗ *Function to generate the synthetic samples.* ∗)
while $N \neq 0$
    Choose a random number between 1 and k, call it nn. This step chooses one of the k nearest neighbors of i.
    for $attr \leftarrow 1$ **to** $numattrs$
      Compute: $dif = Sample[nnarray[nn]][attr] - Sample[i][attr]$
      Compute: gap = random number between 0 and 1
      $Synthetic[newindex][attr] = Sample[i][attr] + gap * dif$
    endfor
    $newindex{+}{+}$
    $N = N - 1$
endwhile
return (∗ *End of Populate.* ∗)

end procedure

The Wilson's Edited Nearest Neighbor Rule (**ENN**) (Wilson, 1972) removes any instance with a class label different from the class of at least two of its three nearest neighbors.

The One-sided selection (**OSS**) (Kubat & Matwin, 1997) is an undersampling method that first applying the CNN to find a consistent subset, and then applying the TLINK to remove noise and borderline examples.

10.3.2 Classifier-Level Approach

One of the methods in the classifier-level approach, which is popularly used to deal with imbalanced data, is **cost-sensitive learning** (**CSL**). To apply this method, we have to explicitly determine the costs. However, "*What are the costs in this case?*". Let us discuss about it in the following.

Most classifiers assume that misclassification costs between the minority and majority classes (false negative and false positive costs) are the same (this leads to the error rate). Nevertheless, in most real-world applications, this assumption may be not true. For example, in customer relationship management, the cost of mailing to non-buyers is less than the cost of not mailing to the buyers (Elkan, 2001); or the cost of misclassifying a non-terrorist as terrorist is much lower than the cost of misclassifying an actual terrorist who could carry a bomb to a flight. Another example is cancer diagnosis: misclassifying a cancer is much more serious than the false alarm since the patients could lose their life because of a late diagnosis and treatment (Sheng & Ling, 2006). Cost is not necessarily monetary, for examples, it can be a waste of time or even the severity of an illness (Domingos, 1999).

Let $C(i, j)$ be the cost of predicting an example belonging to class i when in fact it belongs to class j; the cost matrix is defined in Table 10.2.

Table 10.1: Confusion matrix

		Predict classes	
		Positive	Negative
Actual classes	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

Table 10.2: Cost matrix

		Predict classes	
		Positive	Negative
Actual classes	Positive	$C(+,+)$	$C(-,+)$
	Negative	$C(+,-)$	$C(-,-)$

Given the cost matrix as in Table 10.2, then an example x can be classified into class i with minimum expected cost by using the *Bayes risk* criterion (or *conditional risk*) (Elkan, 2001) as

$$\mathcal{H}(x) = \arg\min_i \left(\sum_{j \in \{-,+\}} P(j|x) C(i,j) \right) \tag{10.1}$$

where $P(j|x)$ is the posterior probability of classifying an example x as the class j.

Usually, we assume that there is no cost for the correct classifications, so the cost matrix can be described by the *cost ratio* as the following

$$costRatio = \frac{C(-,+)}{C(+,-)} \tag{10.2}$$

The purpose of cost-sensitive learning is to build a model that can predict the new examples with minimum misclassification costs. We also call it *total costs*, described as

$$totalCost = C(-,+) \cdot \#FN + C(+,-) \cdot \#FP \tag{10.3}$$

where $\#FN$ and $\#FP$ are the number of false negative and false positive examples, respectively.

One of state-of-the-art methods in cost-sensitive learning is MetaCost (Domingos, 1999). This is a method for making an arbitrary classifier cost-sensitive by wrapping a cost-minimizing procedure around it, using Bayes risk as in equation 10.1. MetaCost treats the underlying classifier as a "black box", requiring no knowledge of its functioning or change to it. Its main idea is to relabel the class of each example x to the class that in the final model gives the lowest total cost. Details of this method is presented in Algorithm 9.

Over the time, many other works based on cost-sensitive learning have been proposed. For example, Ting (1998) introduced an instance-weighting method to induce cost-sensitive trees; two other methods were investigated on cost-sensitive learning with decision trees (Sheng *et al.*, 2005, 2006); while Chai *et al.* (2004) introduced cost-sensitive learning with Naive Bayes to determine how unknown attributes are selected to perform the test on, in order to minimize the sum of the misclassification costs and the test costs.

10.3.3 Combination Approach

Many works in the literature have been focused in this approach. For example, Akbani *et al.* (2004), first, applied the SMOTE (Chawla *et al.*, 2002) to balance the data set, then, built the model using support vector machines with different cost parameters for different classes (Veropoulos *et al.*, 1999). Yan *et al.* (2003) used ensemble learning to deal with class imbalance while Liu *et al.* (2009) combines undersampling with ensemble methods. Tang *et al.* (2009) focused on incorporating different re-balance heuristics

Algorithm 9 MetaCost (Domingos, 1999); where S is the training set; L is the classifier; C is a cost matrix; m is the number of resamples to generate; n is the number of examples in each resample; p is True if L produces class probabilities; q is True if all resamples are to be used for each example.

procedure METACOST(S, L, C, m, n, p, q)

For $i = 1$ to m
  Let S_i be a resample of S with n examples.
  Let M_i = Model produced by applying L to S_i.

For each example x in S
  For each class j
    Let $P(j|x) = \frac{1}{\sum_i 1} \sum_i P(j|x, M_i)$
    Where
      If p then $P(j|x, M_i)$ is produced by M_i
      Else $P(j|x, M_i) = 1$ for the class predicted
        by M_i for x, and 0 for all others.
      If q then i ranges over all M_i
      Else i ranges over all M_i such that $x \notin S_i$.
  Let x's class = $\text{argmin}_i \sum_j P(j|x)C(i, j)$.

Let M = Model produced by applying L to S.

Return M.

end procedure

to support vector machines to tackle the problem of class imbalance while Wang & Japkowicz (2010) and Li *et al.* (2008b) incorporate support vector machines into a boosting method. The other works concentrate on changing the classifier internally, for example support vector machines, to deal with class imbalance such as (Chen *et al.*, 2005; Lessmann, 2004).

Several other methods have also been shown to improve the performance of the classifiers when learning from imbalanced data, e.g. in Cieslak & Chawla (2008); Hido & Kashima (2008); Klement *et al.* (2009); Li & Zhang (2011); Liu *et al.* (2010); Tang *et al.* (2009); Wang & Japkowicz (2010); Wu *et al.* (2007); Yang *et al.* (2011) and more (He & Garcia, 2009).

As already seen, hundreds of papers have been proposed for dealing with class imbalance problem. However, in the scope of this study, we only use the most popular as well as the state-of-the-art methods for comparisons. Those are the SMOTE (Chawla *et al.*, 2002), TLINK (Tomek, 1976), MetaCost (Domingos, 1999), and the foundation of cost-sensitive learning (CSL) (Elkan, 2001).

10.3.4 Evaluation Measures

As mentioned at beginning, to evaluate the models in the case of learning from imbalanced data, accuracy metric is not preferred. Instead, the literature usually use the area under the ROC curve (AUC), the GMean, the F-Measure, etc which are related to some other metrics described in the following.

The True Positive Rate (TPR) (or the Recall R) is the proportion of positive examples correctly classified as belonging to the positive class, determined by:

$$TPR = R = \frac{TP}{TP + FN} \tag{10.4}$$

where TP and FN are described in Table 10.1.

The True Negative Rate (TNR) is the proportion of negative examples correctly classified as belonging to the negative class, determined by:

$$TNR = \frac{TN}{FP + TN} \tag{10.5}$$

The False Positive Rate (FPR) is the proportion of negative examples misclassified as belonging to the positive class, determined by

$$FPR = \frac{FP}{TN + FP} \tag{10.6}$$

The Precision (P) is the positive predictive value, determined by:

$$P = \frac{TP}{TP + FP} \tag{10.7}$$

The F-Measure is an evaluation metric which considers both the Precision and the Recall of the testing results

$$F - Measure = \frac{(1 + \beta^2) \times P \times R}{(\beta^2 \times P + R)} \tag{10.8}$$

where β is used to set equal to 1, and in this case we also call it $F1 - Measure$.

Another metric which is usually used for learning in imbalanced data environment is the GMean (Hido & Kashima, 2008; Kubat & Matwin, 1997; Liu *et al.*, 2009; Wang & Japkowicz, 2010), which balances both the true positive rate and the true negative rate.

$$\text{GMean} = \sqrt{TNR \cdot TPR} = \sqrt{\frac{TN}{FP + TN} \cdot \frac{TP}{TP + FN}} \tag{10.9}$$

The most popular metric which is generally used in machine learning research as well as class imbalance problem is the area under the ROC curve (AUC) Bradley (1997), in which the ROC (Receiver Operating Characteristic) curve is a graphical approach

for displaying the trade-off between the TPR (on the y-axis) and the FPR (on the x-axis) of a classifier.

However, Hand (2006, 2009) has shown that the AUC has a deficiency since it implicitly uses different misclassification cost distributions for different classifiers. Specifically, using the AUC is equivalent to averaging the misclassification loss over a cost ratio distribution which depends on the score distributions. Since the score distributions depend on the classifier itself, employing the AUC as an evaluation measure actually means measuring different classifiers using different metrics. To overcome this incoherence, the "H measure" was proposed, which uses a symmetric Beta distribution to replace the implicit cost weight distribution in the AUC. We briefly summarize the H-Measure in the following (detailed descriptions can be found in Hand (2006, 2009)).

As already denoted in Section 10.3.2, for binary classification problem, the minority class is also labeled 0 and the majority class is labeled as 1. Since we assume that there are no costs for correct classification, the cost matrix in Table 10.2 can also be presented as in Table 10.3

Table 10.3: Simplified cost matrix

		Predict classes	
		Positive	Negative
Actual classes	Positive	0	c_0
	Negative	c_1	0

where c_0 and c_1 are the misclassification costs for class 0 (minority) and class 1 (majority) respectively.

Let $s = s(x)$ be a score produced by the classifier for example x; π_0 and π_1 be the prior probabilities; $f_0(s)$ and $f_1(s)$ be the probability density functions; $F_0(s)$ and $F_1(s)$ be the cumulative distribution functions, for class 0 and class 1 respectively.

With a classification threshold t, one can determine the label for an instance x, e.g., if $s(x) > t$ the example x is classified as class 1, otherwise x is classified as class 0. Thus, choosing a value t will incur a misclassification loss, that misclassification loss is determined by

$$c_1\pi_1(1 - F_1(t)) + c_0\pi_0F_0(t) \tag{10.10}$$

Among many possible values of the threshold t, once can choose an optimal t which can minimize the loss in equation 10.10. The optimal threshold is denoted as $T(c_0, c_1)$ and determined by

$$T(c_0, c_1) \triangleq \arg\min_t \left(c_1\pi_1(1 - F_1(t)) + c_0\pi_0F_0(t)\right) \tag{10.11}$$

Since the optimal threshold depends only on the ratio of the costs, and not on their absolute value (Hand, 2009), we can transform the pair (c_0, c_1) to a pair (b, c), where

$b = c_0 + c_1$ and $c = c_1/(c_0 + c_1)$; Using this pair (b, c), equation (10.11) can be simplified as

$$T(c) = \arg\min_t \left(c\pi_1(1 - F_1(t)) + (1 - c)\pi_0 F_0(t)\right) \tag{10.12}$$

Now, let

$$Q(t; b, c) \triangleq \{c\pi_1(1 - F_1(t)) + (1 - c)\pi_0 F_0(t)\}b$$

be the loss for arbitrary choice of t; and let $v(b, c)$ denote a subjective distribution of likely values of the unknown pair (b, c) (if we can supply precise misclassification costs, v will be a delta function), then the overall expected minimum loss is determined by

$$L = \int_0^1 \int_0^\infty Q(T(c); b, c)v(b, c)dbdc$$
$$= \int_0^1 \{c\pi_1(1 - F_1(T(c))) + (1 - c)\pi_0 F_0(T(c))\}w(c)dc$$

where $w(c) = \int bv(b, c)db$ serves as a weight function over the losses associated with different values of c (different cost ratios) when calculating the overall expected minimum loss. This is also an implicit cost weight function used when calculating the AUC. Hand (2009) proposed to replace this $w(c)$ with an explicit Beta distribution, and now, the general loss becomes:

$$L_{\alpha,\beta} = \int Q(T(c); b, c)u_{\alpha,\beta}(c)dc; \tag{10.13}$$

where

$$u_{\alpha,\beta} = beta(c; \alpha, \beta) = \frac{c^{\alpha-1}(1 - c)^{\beta-1}}{B(1; \alpha, \beta)}$$

and the maximum loss is determined by

$$L_{Max} = \pi_1 \int_0^{\pi_0} cu_{\alpha,\beta}(c)dc + \pi_0 \int_{\pi_0}^1 (1 - c)u_{\alpha,\beta}(c)dc$$

The H measure is determined by subtracting from 1 to the overall loss ratio:

$$H = 1 - \frac{L_{\alpha,\beta}}{L_{Max}} = 1 - \frac{\int Q(T(c); b, c)u_{\alpha,\beta}(c)dc}{\pi_1 \int_0^{\pi_0} cu_{\alpha,\beta}(c)dc + \pi_0 \int_{\pi_0}^1 (1 - c)u_{\alpha,\beta}(c)dc}$$

Hand (2009) proposed using a symmetric Beta function ($beta(c; 2, 2)$) for the values of $u_{\alpha,\beta}(c)$, the actual H value is now determined by

$$H = 1 - \frac{\int Q(T(c); b, c)\ beta(c; 2, 2)dc}{\pi_1 \int_0^{\pi_0} c\ beta(c; 2, 2)dc + \pi_0 \int_{\pi_0}^1 (1 - c)\ beta(c; 2, 2)dc} \tag{10.14}$$

However, as we discussed at the beginning, when learning from imbalanced data, misclassifying a minority class example (e.g., a fraud credit card transaction) is much more serious than misclassifying a majority example. We propose using an *asymmetric* Beta distribution such as $beta(x; 4, 2)$ instead of the symmetric one as in the H measure. We will describe more details about this in Section 10.6.

10.4 Compound Cost-Sensitive Learning Methods

In this section, based on the cost-sensitive learning and the sampling methods we propose the compound methods for learning from imbalanced data. In the first method, we re-balance the data sets by using one of the sampling methods as described in Section 10.3.1, we then apply the cost-sensitive learning on those sampling data. We call it the compound of sampling technique with cost-sensitive learning, named as **S-CSL**.

One problem in cost-sensitive learning is that we do not know exactly what are the costs on the given data sets since they depend on the application areas. In the literature, they usually assume that they have some cost ratios at hand, then building and testing the models using those cost ratios. Finally, results are reported by averaging the misclassification costs over those ratios, e.g., in (Domingos, 1999; Liu & Zhou, 2006; Ting, 2002). However, in the second proposed method, based on the idea from hyper parameter learning, we propose using cost-sensitive learning by optimizing the cost ratio instead of averaging on several ratios as in the literature, or as in the first proposed method. We call this method **O-CSL**[1].

10.4.1 Combining Sampling Techniques with CSL (S-CSL)

In this method, we mainly use Support Vector Machines (SVM) (Cortes & Vapnik, 1995) as a base classifier. Let us briefly summarize some notations that we have used in the SVM.

Given a dataset $\mathcal{D}$ consisting of n examples (x_i, y_i), where $x_i \in \mathcal{X}$ are input features and y_i is the target class, $y_i \in \{-1, +1\}$. The SVM predicts a new example x by

$$f(x) = sign\left(\sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}, \mathbf{x_i}) + b\right) \tag{10.15}$$

where $k(\mathbf{x}, \mathbf{x_i})$ is a kernel function, b is the bias, and α_i is determined by solving the Lagrangian optimization problem

$$L_p = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i}^{n}\xi_i - \sum_{i}^{n}\alpha_i\{y_i(x_i.w + b) - 1 + \xi_i\} - \sum_{i}^{n}\mu_i\xi_i \tag{10.16}$$

[1]This method was called CSL-OCRL in Thai-Nghe *et al.* (2010d)

where ξ_i is a slack variable, μ_i is a Lagrange multiplier, and C is a user-specified hyper parameter representing the penalty of misclassifying the training instances.

For non-linear problems, the kernel k is used to maximize margin hyperplanes. Two commonly used kernel functions are the polynomial kernel

$$k(\mathbf{x}, \mathbf{x_i}) = (\gamma \mathbf{x} \cdot \mathbf{x_i} + r)^p \tag{10.17}$$

and the radial basis function (RBF) kernel

$$k(\mathbf{x}, \mathbf{x_i}) = e^{-\gamma \|\mathbf{x} - \mathbf{x_i}\|^2} \tag{10.18}$$

To determine the hyper parameter C, exponent p, and γ in equations (10.16), (10.17), and (10.18) of the SVM, we applied grid search as described in Algorithm 10. First, a "raw search" on the powers of two (e.g. $2^{-15} \dots 2^{10}$ for C values) was used to identify a good region, then a "smooth search" around that region was conducted (Hsu *et al.*, 2003).

Algorithm 10 Hyper parameter search for optimizing metric E with step δ for C value, and step λ for γ value in RBF kernel

procedure HYPERSEARCH($\mathcal{D}_{train}, E, \delta, \lambda$)
returns the best hyperparameters Θ *for eval. metric* E
  $(\mathcal{D}_{LocalTrain}, \mathcal{D}_{Val}) \leftarrow \mathcal{D}_{train}$ //split for 5-fold CV
  //Raw search:
  $bestC, best\gamma \leftarrow 0$
  for $i \leftarrow -15, \dots, 10$ **do**
    for $j \leftarrow -15, \dots, 0$ **do**
      $\gamma \leftarrow 2^j$; $C \leftarrow 2^i$
      $buildLocalSVM(\mathcal{D}_{LocalTrain}, \gamma, C)$
      $TestLocalModel(\mathcal{D}_{Val})$ //using metric E
      Update $bestC, best\gamma$
    end for
  end for
  //Smooth search:
  for $i \leftarrow bestC - 1, \dots, bestC + 1, step\ \delta$ **do**
    for $j \leftarrow best\gamma - 0.1, \dots, best\gamma + 0.1, step\ \lambda$ **do**
      $\gamma \leftarrow j$; $C \leftarrow i$
      $buildLocalSVM(\mathcal{D}_{LocalTrain}, \gamma, C)$
      $TestLocalModel(\mathcal{D}_{Val})$ //using metric E
      $\Theta \leftarrow C, \gamma$ //Update the best parameter values
    end for
  end for
  return Θ
end procedure

For sampling methods, since each data set has its own structure, the percentages of undersampling and oversampling are also different. In our work, these percentages are also treated as hyper parameters. For oversampling, we search on the percentages from 50, 100, 150, ... to a balanced distribution between the two classes. Similarly for undersampling, we also search on the percentages from 10, 20, 30, ... to a balanced distribution.

Algorithm 11 Compound Cost-Sensitive Learning (S-CSL)

1: **procedure** S-CSL($\mathcal{D}, \mu, \mathcal{C}$)
Input: Dataset $\mathcal{D}$, sampling method μ, and cost matrix $\mathcal{C}$
Output: Label for new example x^*
2: $(\mathcal{D}_{train}, \mathcal{D}_{test}) \leftarrow \mathcal{D}$ //split for 5-fold CV
3: **for** each Φ in {sample space percentages} **do**
4: $\mathcal{D}_{Tr\mu\Phi} \leftarrow GenerateDistribution(\mathcal{D}_{train}, \mu, \Phi)$
5: $\Theta_{\mu\Phi} \leftarrow HyperSearch(\mathcal{D}_{Tr\mu\Phi}, TC, 0.25, 0.01)$
//0.25 and 0.01 are increase-step of C and γ in RBF kernel
6: $\Theta^*_{\mu\Phi} \leftarrow$ Update-best-hyperparameters for $\mathcal{D}^*_{Tr\mu\Phi}$
7: **end for**
8: //Train SVM model with parameters $\Theta^*_{\mu\Phi}$ on $\mathcal{D}^*_{Tr\mu\Phi}$

$$f(x) \leftarrow \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}, \mathbf{x_i}) + b$$

9: //Fitting a sigmoid function to SVM outputs to get the posterior probability.

$$P(j|x) \leftarrow \frac{1}{1 + e^{\alpha f(x) + \beta}}$$

10: //Testing example x^* in $\mathcal{D}_{test}$

$$\mathcal{H}(x^*) \leftarrow \arg\min_i \left(\sum_{j \in \{-1,+1\}} P(j|x^*) \mathcal{C}_{ij} \right)$$

11: **end procedure**

For S-CSL, we propose combining one of four sampling techniques with CSL. These sampling techniques include non-heuristic under-/over-sampling (RUS, ROS) and heuristic under-/over-sampling (TLINK, SMOTE).

Details of the S-CSL are described in Algorithm 11. In the first step, we divide the original data set into two separate train and test sets; then, either one of four sampling techniques $\mu \in$ {RUS, TLINK, ROS, SMOTE} with different sampling percentages Φ is applied on the train set to generate new distributions; next, we perform the hyper parameter search (see Algorithm 10) on the new training sets to determine the best

parameters in terms of total costs; in the next step, the SVM is built based on the best hyper parameters found. Outputs of the SVM are fitted by a sigmoid function[1] to get the posterior probabilities; finally, we use the Bayes risk criterion (equation 10.1) to predict new examples in the test set.

Since most data sets do not have the cost ratios, so we assumed cost ratios from the set $\{2^2, 2^4, 2^6, 2^8\}$. The final results will be obtained by averaging the misclassification costs on those ratios, as used in the literature (Domingos, 1999; Liu & Zhou, 2006; Ting, 2002).

Algorithm 12 Locally optimize the cost ratio with step length η

procedure OPTIMIZECOSTRATIO($\mathcal{D}_{train}, \Theta, \eta$)
Input: $\mathcal{D}_{train}$, SVM parameters Θ, step length η
Outputs: the best cost ratio for GMean
  $(\mathcal{D}_{LocalTrain}, \mathcal{D}_{Val}) \leftarrow \mathcal{D}_{train}$ ▷ split for 5-fold CV
  $ImbaRatio \leftarrow \frac{|Major|}{|Minor|}$ ▷ imbalance ratio of $\mathcal{D}_{train}$
  $maxRatio \leftarrow ImbaRatio * 1.5$
  $curRatio \leftarrow 1.0$
  $bestGMean \leftarrow 0$
  $buildLocalModel(\mathcal{D}_{LocalTrain}, \Theta)$
  while $curRatio <= maxRatio$ **do**
    $curGMean \leftarrow testLocalModel(\mathcal{D}_{Val}, curRatio)$
    if $(curGMean > bestGMean)$ **then**
      $bestGMean \leftarrow curGMean$
      $bestCostRatio \leftarrow curRatio$
    end if
    $curRatio \leftarrow curRatio + \eta$
  end while
  return $bestCostRatio$
end procedure

10.4.2 Optimized Cost Ratio for CSL (O-CSL)

In the first method S-CSL, we have assumed that we have several cost ratios then trying them and taking the average results. In this section, we will introduce a method that may supply a good cost ratio to the classifiers.

As discussed in the literature, the cost ratio was determined by inverting the prior distributions (Margineantu, 2000; Raskutti & Kowalczyk, 2004). For example,

$$costRatio = \frac{\text{the number of majority examples}}{\text{the number of minority examples}}$$

[1]The sigmoid function has two parameters: α and β. These values can be determined by using maximum likelihood (Platt, 1999b), but for straightforward, we set them to 1

However, this choice leads to the Kolmogorov-Smirnov statistic being the performance metric (Hand, 2009). Hand (2009) said that this is almost certainly inappropriate, precisely because it is made not on the basis of consideration of the relative severity of the misclassifications in the presenting problem, but simply on grounds of convenience (Hand, 2006, 2009). In our method, O-CSL, we treat this cost ratio as a hyper parameter, and locally optimize this hyper parameter.

Algorithm 13 Optimized Cost Ratios for CSL (O-CSL)

1: **procedure** O-CSL($\mathcal{D}$)
Input: Dataset $\mathcal{D}$
Output: Label for new example x^*
2: $(\mathcal{D}_{train}, \mathcal{D}_{test}) \leftarrow \mathcal{D}$ //split for 5-fold CV
3: $\Theta \leftarrow HyperSearch(\mathcal{D}_{train}, GMean, 0.25, 0.01)$
4: //Optimize locally with increase-step 0.25 for cost ratio

$$C^*(i,j) \leftarrow OptimizeCostRatio(\mathcal{D}_{train}, \Theta, 0.25)$$

5: //Train SVM model with parameters Θ on $\mathcal{D}_{train}$

$$f(x) \leftarrow \sum_{i=1}^{n} \alpha_i y_i k(\mathbf{x}, \mathbf{x_i}) + b$$

6: //Fitting a sigmoid function to SVM outputs to get the posterior probability:

$$P(j|x) \leftarrow \frac{1}{1 + e^{\alpha f(x) + \beta}}$$

7: //Testing example x^* in $\mathcal{D}_{test}$:

$$\mathcal{H}(x^*) \leftarrow \arg\min_i \left(\sum_{j \in \{-1,+1\}} P(j|x^*) C^*(i,j) \right)$$

8: **end procedure**

Algorithm 12 shows how we can obtain the cost ratio. Please note that in line 4, we use this kind of search because the data sets in this study are not extremely imbalanced and our preliminary experiments showed that the results are not significantly improved (in terms of the GMean metric) when using a high cost ratio. Also, We used the GMean as an evaluation metric in this study because previous works show that GMean is more appropriate in the case of imbalanced data (Hido & Kashima, 2008; Kubat & Matwin, 1997; Liu *et al.*, 2009; Wang & Japkowicz, 2010). However, one can use any other metric such as the AUC, F-Measure, etc in the same manner.

Algorithm 13 presents the details of the O-CSL method. The differences between

the O-CSL and the S-CSL are that the O-CSL learns on the original data instead of resampling data and the O-CSL uses the optimized cost ratio instead of using several assumed cost ratios.

10.5 Thresholding on Bayesian Posterior Probabilities

As described in the literature (Chawla *et al.*, 2004; He & Garcia, 2009), instead of categorizing the methods as "Data-level", "Classifier-level" and "Combination-level"as in Section 10.3, we can also categorize the methods into three groups in different perspectives: Pre-processing, internal classifier processing, and post-processing. The pre-processing group is actually the "Data-level" group, which related to (re)sampling methods. The internal classifier processing group is also known as "Classifier-level", in which the algorithms are manipulated for different classifiers, such as for SVM (Veropoulos *et al.*, 1999), for C4.5 (Liu *et al.*, 2010), for ensemble learning (Liu *et al.*, 2009), and so on (He & Garcia, 2009).

The last group is post-processing, which mainly relies on the posterior probabilities produced by the classifiers. In this group, most of the literature have applied cost-sensitive learning to "wrap around" the classifiers with a Bayes risk function (as in equation 10.1), e.g. to C4.5 (Domingos, 1999; Elkan, 2001), Naive Bayes (NB) (Sheng *et al.*, 2006), and support vector machines (Thai-Nghe *et al.*, 2010d). Since the methods in this group are based on posterior probabilities, it is important to know that which classifiers that one choose should produce good posterior probabilities. Moreover, as discussed in (Fan *et al.*, 2005; Hido & Kashima, 2008), averaging on probabilities can produce better results than on voting the majority. Bayesian Networks (BN) is a good candidate for this choice. As far as we know, almost the literatures have focused on using C4.5, and Naive Bayes which has a strong assumption on the independence among attributes given the target attribute. When relaxing on this assumption, one can get the better results (Cheng & Greiner, 1999; Friedman *et al.*, 1997).

Moreover, previous works have tuned the decision threshold by some different ways. For examples, Maloof (2003) has experimented on the moving of decision threshold of the ROC curve and adjusted the cost matrix to deal with unbalanced and unknown cost data. The authors compared the results of C5.0, k-Nearest Neighbors, and Naive Bayes; In Sheng & Ling (2006), the authors used a method varied from Elkan (2001) called Thresholding. This method used C4.5 as a base classifier and selected a proper threshold from training instances according to the misclassification costs. The results are reported in term of total cost and took the misclassification costs into account. Klement *et al.* (2009) proposed an ensemble of Naive Bayes classifiers with an adjusted decision threshold trained on random undersampling data to deal with class imbalance.

In this study, we propose utilizing the Bayesian posterior probabilities to deal with imbalanced data. Concretely, we locally optimize the decision threshold on the posterior probabilities produced by several Bayesian Networks (e.g. general BN, TAN, BAN, or Markov Blanket structure, which we will introduce in next section). Once the optimal

threshold is archived, we use it for the final classification. We also apply the same way on resampling data, instead of on the original one. Different from the Thresholding method Sheng & Ling (2006) which has to know the cost matrix (or at least the cost ratio) before learning process, the proposed methods do not require the cost ratios.

10.5.1 Learning in Bayesian Networks

Since the methods in this section are based on Bayesian Networks, we will briefly summarize it in this section.

Bayesian Networks (BN) are defined by a pair $B = \langle G, \Theta \rangle$, where G is the directed acyclic graph with a set of nodes $X = (x_1, x_2, \ldots, x_n)$ represent random variables, and edges represent the direct dependencies between these variables, and Θ is a set of parameters of the network (Friedman *et al.*, 1997).

Naive Bayes **(NB)** is a type of BN which has assumptions that all the variables are conditionally independent given the class variable and are directly dependent on the class variable, as in Figure 10.1a[1].

Tree Augmented Naive Bayes **(TAN)** (Friedman *et al.*, 1997) relaxes the assumption in NB by allowing arcs between the children of the target node, as in Figure 10.1b

Bayesian Network Augmented Naive Bayes **(BAN)** (Cheng & Greiner, 1999) is a BN which all other nodes are children of the target node, but a complete BN is constructed between the child nodes rather than just a tree as in TAN, as in Figure 10.1c.

The Markov Blanket Bayesian Classifier **(MB)** (Madden, 2002) is a BN which has Markov Blanket property at a target node. The Markov Blanket for a node in BN consists of its parents, its children, and the parents of its children, as in Figure 10.1d.

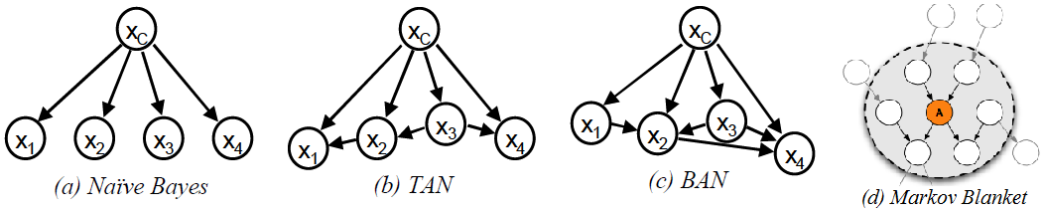


Figure 10.1: Bayesian Network types

Similar to the other literature (Cheng & Greiner, 1999; Friedman *et al.*, 1997; Hall *et al.*, 2009; Madden, 2002), this study focuses on the discrete and non-missing value variables[2]. The learning tasks in BN consist of two steps. The first step is to learn the network structure and the second step is to compute the conditional probability tables (CPTs).

[1] Picture source: (Madden, 2002) and Wikipedia(en.wikipedia.org/wiki/File:MarkovBlanket.png)

[2] We can discretize the numeric attributes, and replace all missing values for nominal and numeric attributes with the modes and means, respectively.

To learn the structure B_S of the BN, we consider it as an optimization problem Hall *et al.* (2009) and need to maximize the quality measure Q of B_S given dataset D. In this study, we use the Bayesian metric as a quality measure, determined by the following Eq:

$$Q(B_S|D) = P(B_S)\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(N'_{ijk}+N_{ijk})}{\Gamma(N'_{ijk})} \tag{10.19}$$

where $P(B_S)$ is prior probability of B_S; n is the number of variables; Γ is a Gamma function; r_i and q_i are the cardinality of node x_i and a set of its parents Π_i, respectively; N_{ij} is $|D|$ for which Π_i takes its jth value; N_{ijk} is $|D|$ for which Π_i takes its jth value and for which x_i takes its kth value; $N_{ij} = \sum_{k=1}^{r_i} N_{ijk}$; N'_{ij} and N'_{ijk} represent choices of priors on counts restricted by $N'_{ij} = \sum_{k=1}^{r_i} N'_{ijk}$ (Hall *et al.*, 2009). Since $P(B_S)$ is constant, to maximize Q, we just need to maximize the second inner product in equation (10.19) as the following

$$\Psi = \prod_{j=1}^{q_i}\frac{\Gamma(N'_{ij})}{\Gamma(N'_{ij}+N_{ij})}\prod_{k=1}^{r_i}\frac{\Gamma(N'_{ijk}+N_{ijk})}{\Gamma(N'_{ijk})} \tag{10.20}$$

To do this, we use K2 algorithm Cooper & Herskovits (1992) which initially assumes that a node has no parents, and then adding incrementally its parent that can increase the probability of the resulting network. This process repeats greedily until the addition of the parent does not increase the network structure probability. Concretely, each iteration of K2, an arc is added to node u from the node v that maximizes $\Psi(u, \Pi_u \cup v)$, where Π_u is the set of parents of node u. If $\Psi(u, \Pi_u) > \Psi(u, \Pi_u \cup v)$ then no arc is added (Madden, 2002).

After network structure is learned, we can estimate the CPTs by:

$$P(x_i = k|\Pi_{x_i} = j) = \frac{N_{ijk} + N'_{ijk}}{N_{ij} + N'_{ij}} \tag{10.21}$$

Once having the CPTs, one can infer for any new event. The probability of an arbitrary event $\mathrm{X} = (x_1, x_2, \ldots, x_n)$ is determined by

$$\mathcal{P}(X) = \mathcal{P}(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n}\mathcal{P}\left(x_i \,\middle|\, \Pi_{x_i}\right) \tag{10.22}$$

Given a dataset D consists of a class variable y and a set of attribute variables $X = (x_1, x_2, \ldots, x_n)$, we can infer the class value for y by calculating the $\arg\max_y(\mathcal{P}(y|X))$ from the probability distribution in equation (10.22).

10.5.2 Thresholding on Bayesian Posterior Probabilities

We now describe two methods which utilizing the Bayesian posterior probabilities. In the first method, we optimize the threshold θ on the validation set to maximize the

metric (e.g. F1-Measure) Once having an optimal θ, we use it for the final classifier. We test this method on several Bayesian Network types, such as the general BN, TAN, BAN, and Markov Blanket structure which we call them **BNOpt**, **TANOpt**, **BANOpt**, and **MBOpt** respectively.

Details of this method is described in Algorithm 14, which we call *Learn-BNOpt*. For each BN type Φ, we optimize its threshold as in line 2 and Algorithm 15. The next steps are to learn the structure of that BN and compute the CPTs as in line 3. Once the optimal threshold is found and CPTs is constructed, we can use them for inferring the new examples as in line 4. The indicator function $\mathcal{I}(.)$ gives the positive class if the expression is true, and negative class for the inverse.

Algorithm 14 Learning optimal threshold for Bayesian Networks

1: **procedure** LEARN-BNOPT($\mathcal{D}_{train}, \mathcal{D}_{test}, \Phi$)
Input: $\mathcal{D}_{train}$ *with* $x_i \in \mathcal{X}$ *attributes, target* $y_i \in \{-1, +1\}$
$\Phi \in \{$*BN, TAN, BAN, Markov Blanket BN*$\}$
Outputs: Label for new example x^* *in* $\mathcal{D}_{test}$
2: $\theta \leftarrow OptimizeThreshold(\mathcal{D}_{train}, \Phi)$
3: Learn the structure and CPTs of Φ as in equation (10.19,10.20,10.21)

$$\mathcal{P}(x_1, x_2, \ldots, x_n) \leftarrow \left(\prod_{i=1}^{n} \mathcal{P}\big(x_i \,\big|\, \Pi_i\big); x_i \in \mathcal{D}_{train}; \Phi \right)$$

4: Test for new example x^* from $\mathcal{D}_{test}$

$$\mathcal{H}(x^*) \leftarrow \mathcal{I}\big(\mathcal{P}(j = +1|x^*) > \theta\big)$$

5: **end procedure**

Algorithm 15 describes how to get the optimal threshold. We do 5-fold cross validation[1] to get the average results (lines 2-7). Next, from the average score outputs, we get the unique values for the minority class (line 8). We consider each score value as a threshold and re-calculate the F1-Measure then update the one which has the maximal value (line 9-16). We can do another way by treating this threshold as a hyper parameter and do the hyper parameter search as in Algorithm 10, however, that way needs more running times than the current one.

The second method is called *Learn-EnsBNOpt*, which learns an ensemble classifier of several BNs, as described in Algorithm 16. Specifically, we train each classifier Φ and get its model $\mathcal{M}_\Phi$ respectively, as in lines 2-4. We combine these models by averaging on the probabilities as in line 5[2]. The reason for this combination is that it is well-

[1] Of course, one can use any other number of folds for this, but for consistence with the main procedure we use 5-fold cross-validation

[2] We use the "Vote" method in (Hall *et al.*, 2009). The interesting readers can see (Kittler *et al.*, 1998; Kuncheva, 2004) for more details about how to combine the models.

Algorithm 15 Optimize the threshold on the validation set

1: **procedure** OPTIMIZETHRESHOLD($\mathcal{D}_{train}, \Phi$)
Input: $\mathcal{D}_{train}$, Classifier Φ (BNs)
Outputs: the best threshold for F1Measure
2: **for** $i \leftarrow 1, \ldots, 5$ **do**
3: $(\mathcal{D}^i_{LocalTrain}, \mathcal{D}^i_{Val}) \leftarrow \mathcal{D}_{train}$ ▷ split for 5-fold cross-validation
4: $\mathcal{M} \leftarrow buildLocalModel(\mathcal{D}^i_{LocalTrain}, \Phi)$
5: $score^i \leftarrow testLocalModel(\mathcal{M}, \mathcal{D}^i_{Val}, \Phi)$
6: **end for**
7: $predictionScores \leftarrow AVG(score^i)$ ▷ average on 5-fold CV
8: $UniqueScores \leftarrow$ Get unique values from $predictionScores$
9: $bestF1Measure \leftarrow 0$; $bestThreshold \leftarrow 0$
10: **for each** $curThreshold \in UniqueScores$ **do**
11: $currentF1Measure \leftarrow$ Calculate *F1Measure* based on $curThreshold$
12: **if** ($currentF1Measure > bestF1Measure$) **then**
13: $bestF1Measure \leftarrow currentF1Measure$
14: $bestThreshold \leftarrow curThreshold$
15: **end if**
16: **end for**
17: **return** $bestThreshold$
18: **end procedure**

known that an ensemble on several models can work better than any best single model. Moreover, as discussed in (Fan *et al.*, 2005; Hido & Kashima, 2008), averaging on the probabilities can give the result better than on the label voting. In the next step, we learn the optimal threshold on the aggregated model the same as in the first method. Finally, we predict the new examples in the test set as in line 7, where $\mathcal{P}(j|x)$ is the posterior probability of class j given example x in the aggregated model.

10.5.3 Thresholding on Sampling Data

Similar to Algorithm 14, however, instead of learning on the original data, We first rebalance the data sets by using one of the sampling techniques (e.g. ROS, RUS, SMOTE, or TLINK described in Section 10.3.1). We then locally optimize the threshold θ (e.g. to maximize the F1-Measure) Once having an optimal θ, we use it for the final classification.

In another approach, instead of optimizing the threshold, we use the cost threshold as in Lemma 1 and build the models on original data. This method is called **BNCost**. Difference from (Elkan, 2001), we do not take "profit" into account, and we assume that there are no costs (or profits) for correct classifications. The cost threshold is introduced as in the following.

Algorithm 16 Learning optimal threshold on ensemble of Bayesian Networks

1: **procedure** LEARN-ENSBNOPT($\mathcal{D}_{train}, \mathcal{D}_{test}$)
Input: $\mathcal{D}_{train}$ *with* $x_i \in \mathcal{X}$ *attributes, target* $y_i \in \{-1, +1\}$
Outputs: Label for new example x^* *in* $\mathcal{D}_{test}$
2: **for each** $\Phi \in \{$*BN, TAN, BAN, Markov Blanket BN*$\}$ **do**
3: Learn the structure and CPTs of Φ as in equation (10.19,10.20,10.21), get model $\mathcal{M}_\Phi$

$$\mathcal{P}(x_1, x_2, \ldots, x_n) \leftarrow \left(\prod_{i=1}^{n} \mathcal{P}\big(x_i \mid \Pi_i\big); x_i \in \mathcal{D}_{train}; \Phi \right)$$

4: **end for**
5: Combine all $\mathcal{M}_\Phi$ to aggregated model $\mathcal{M}$ by average of probabilities
6: $\theta \leftarrow$ Optimize threshold for aggregated model $\mathcal{M}$
7: Test for new example x^* from $\mathcal{D}_{test}$

$$\mathcal{H}(x^*) \leftarrow \mathcal{I}\big(\mathcal{P}(j = +1|x^*) > \theta\big)$$

8: **end procedure**

Lemma 1: *Given the cost matrix (or cost ratio), a threshold for the classifier can be determined by* $\theta = \frac{C(+,-)}{C(+,-)+C(-,+)}$ *and a new example can be classified as positive if* $P(+|x) > \theta$. *(This is similar to the threshold for Decision Tree induction described in Tan* et al. *(2005)).*

Proof: Using Bayes risk in equation 10.1, let

$$\mathcal{R}(i|x) = \sum_{j \in \{-,+\}} P(j|x)C(i,j) \tag{10.23}$$

To classify as positive class, an example x should be satisfied

$$\mathcal{R}(+|x) < \mathcal{R}(-|x)$$

substitute this expression by the equation 10.23, we have

$$P(-|x)C(+,-) + P(+|x)C(+,+) < P(-|x)C(-,-) + P(+|x)C(-,+)$$

Since there are no costs for correct classifications, $C(+,+) = 0$ and $C(-,-) = 0$. This reduced to

$$P(-|x)C(+,-) < P(+|x)C(-,+)$$

equivalent to

$$(1 - P(+|x))C(+,-) < P(+|x)C(-,+)$$

thus, lead to

$$P(+|x) > \frac{C(+,-)}{C(+,-) + C(-,+)}$$

or $P(+|x) > \theta$ ■

Algorithm 17 Learning BNs with optimal threshold on sampling data

1: **procedure** LEARN-S-BNOPT($\mathcal{D}_{train}, \mathcal{D}_{test}, \Phi, \mu, \Theta$)
Input: $\mathcal{D}_{train}$ with $x_i \in \mathcal{X}$, target $y_i \in \{-1, +1\}$
$\Phi \in$ {BN, TAN, BAN, BN Markov Blanket}
$\mu \in$ {SMOTE, TLINK, ROS, RUS}
$\Theta \in$ {Sampling percentages}
Outputs: Label for new example x^* in $\mathcal{D}_{test}$
2: **for each** $\delta \in \Theta$ **do**
3: $\mathcal{D}_{train\mu\delta} \leftarrow GenerateDistribution(\mathcal{D}_{train}, \mu, \delta)$
4: $\theta \leftarrow OptimizeThreshold(\mathcal{D}_{Train\mu\delta}, \Phi)$
5: $\theta^* \leftarrow$ Update the best value of $\mathcal{D}^*_{Train\mu\delta}$
6: **end for**
7: Learn BN structures and CPTs as in Eq. (10.19, 10.20, 10.21)

$$\mathcal{P}(x_1, x_2, \ldots, x_n) \leftarrow \left(\prod_{i=1}^{n} \mathcal{P}\big(x_i \,\big|\, \Pi_i\big); x_i \in \mathcal{D}^*_{Train\mu\delta}; \Phi \right)$$

8: Test for new example x^* from $\mathcal{D}_{test}$

$$\mathcal{H}(x^*) \leftarrow \mathcal{I}\big(\mathcal{P}(j = +1|x^*) > \theta^*\big)$$

9: **end procedure**

The proposed methods are formulated in Algorithm 17, which we call *Learn-S-BNOpt*. At first, we apply sampling technique on the train set to generate new datasets with more balanced distributions; then, we optimize the threshold of BN Φ to maximize the F1-measure as in lines 2-6. The optimal threshold θ^* and $\mathcal{D}^*_{Train\mu\delta}$ are recorded. The next steps are to learn the structure of that BN and compute the CPTs as in line 7. Once the CPTs are constructed, we can use them for inferring new examples in the test set as in line 8. The indicator function $\mathcal{I}(.)$ gives the positive class if the expression is true and negative class for the inverse.

The θ^* value can be the optimal threshold in the first method or the cost threshold (Lemma 1) in the second method. Please note that in the second method, most datasets do not have the cost matrix (or cost ratio), So, we apply the method in Section 10.4.2 to search for the best ratio.

10.6 B42 - A New Evaluation Measure

As we already pointed out the importance role of learning from imbalanced data in the introduction section. Now, let use give another example, e.g. in terrorist detection at an airport. Suppose that a data set has 999,999 normal passengers (non-terrorist) and only 1 passenger is (or just "looks like") a terrorist. To maximize the accuracy, in this case, all supervised classifiers may classify every passenger as belonging to the non-terrorist ones to get 99.9999% accuracy. However, the most important passenger, the terrorist one, is already misclassified. Obviously, to evaluate the classifiers in this case, the accuracy metric becomes useless, and the GMean and/or the AUC are commonly used instead.

As presented in Section 10.3.4, the AUC has incoherence and the H measure has been proposed to overcome that problem. However, the H measure used a symmetric distribution which is more appropriate for balanced data sets. Furthermore, as investigated in He & Garcia (2009), there is an open issue for the future research in learning from imbalanced data is that the need for a *standardized evaluation* protocol. In this section, we will propose a new evaluation measure which is based on the H measure but more suitable for learning in imbalanced data environment. First, let us review the Beta distributions in the following.

Beta distributions are a popular model for random variables (Degroot & Schervish, 2002) with values in the interval [0,1]. The Beta function, also known as Euler's Beta integral (Degroot & Schervish, 2002), is defined as

$$B(1;\alpha,\beta) = \int_0^1 c^{\alpha-1}(1-c)^{\beta-1}dc.$$

It can also be defined by using the Gamma function

$$B(1;\alpha,\beta) = \frac{\Gamma(\alpha)\,\Gamma(\beta)}{\Gamma(\alpha+\beta)}.$$

A generalization of the Beta function is the incomplete Beta function:

$$B(x;\alpha,\beta) = \int_0^x c^{\alpha-1}(1-c)^{\beta-1}dc.$$

The probability density function of the Beta distribution has its mode at

$$\frac{\alpha-1}{\alpha+\beta-2}$$

and is determined by

$$f(x;\alpha,\beta) = \frac{1}{\mathrm{B}(1;\alpha,\beta)}\,x^{\alpha-1}(1-x)^{\beta-1}$$

$$= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\,\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}.$$

As discussed in Hand (2009), the alternative cost distribution, which can replace the implicit cost weight distribution in the AUC, needs to be a non-uniform one. Thus, an asymmetric Beta distribution would be a good choice for this replacement.

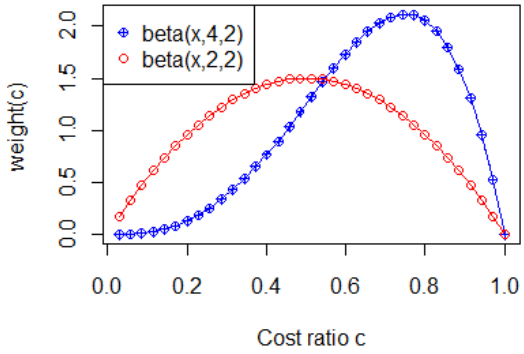


Figure 10.2: Symmetric and Asymmetric Beta Distributions

As we can see in Figure 10.2, for two balanced classes, a symmetric Beta distribution acts as a cost weight distribution, which places most probabilities at 0.5, is used in the H.

However, as we already seen, misclassifying a minority class example (e.g., in terrorist detection system, misclassifying a terrorist who can carry a bomb on a flight) is much more serious than misclassifying a majority class example (e.g, misclassifying a normal passenger as a terrorist). Thus, the misclassification cost c_1 (false negative cost) of the minority is much higher than the misclassification cost c_0 (false positive cost) of the majority, therefore, the cost ratio $c = c_1/(c_0 + c_1)$ should be higher than 0.5. For the aforementioned reason, we use the *asymmetric* Beta distribution B42 as a cost weight distribution. B42 *places higher weight on minority class examples* and is a unimodal distribution with mode at 0.75.

Please note that one can choose some other values for α (e.g., *beta(x,6,2)*, *beta(x,8,2)*...). In those cases, the absolute values of the metrics can be higher, but the relative values are not significantly different. Thus, we decide to use *beta(x,4,2)*, and we call it **B42**.

10.7 Experiments

In this section, we first describe the data sets which are used for experiments. We then present the experimental setting followed by the experimental results.

10.7.1 Data Sets

As we discussed at the beginning, the purpose of this chapter is to provide the methods and the evaluation measure for learning from imbalanced data in general environments rather than only in educational data mining. Thus, we mainly use the data sets from the other domains such as network intrusion detection (KDD Cup 1999 data sets), customer retention rate detection (KDD Cup 2010 data sets), etc. However, we also use the educational data sets for a few experiments.

Educational data sets

We use the Algebra and Bridge data sets as described in Chapter 3. We only select the first 10,000 instances in each data set and use two features: StudentAVG and Solving-StepAVG which are converted to averages as described in Section 3.3.

Other data sets

For the other data sets, we have experimented on both small and large data sets collected from the UCI repository[1] and the Netflix Prize[2]. We group them into 3 groups as in Table 10.4.

All nominal attributes are converted to binary numeric attributes. For multi-class data sets, many of them (e.g., RCV1, News20, etc.) were already transformed to binary-class data sets as in the LIBSVM data set library[3]. The remaining multi-class datasets are converted to binary-class using one-versus-the-rest. We encoded the class which has the smallest number of examples as the minority class, and the rest as the majority one.

The Netflix (nf) data set originally has 100,480,507 ratings from 480,189 customers for 17,770 movies. To create a binary matrix, in which rows represent users/customers and columns represent items/movies, we assign 1 for each observed rating, and 0 otherwise. We then sort the columns based on their class distributions as in Figure 10.3. To create a data set, we choose one column (movie) to be the target, whereas the other columns represent the input features. This way, we can generate 17,770 different data sets. For example, the data set "nf-05p" means that we choose a target column which has 0.5% minority.

Please note that the last five data sets in Table 10.4 are not imbalanced. We use them to see how the results are affected when learning from "nearly balanced" to "highly imbalanced" class distributions.

[1] http://archive.ics.uci.edu/ml/

[2] http://www.netflixprize.com

[3] http://www.csie.ntu.edu.tw/∼cjlin/libsvmtools/datasets/

Table 10.4: Data sets

Dataset	#Examples	#Attributes	#Minority	%Minority	Size
Netflix - 001p	480,189	17,770	52	0.01	2.6 GB
KDD Cup1999 - u2r	4,898,432	123	52	0.001	743.0 MB
KDD Cup 1999 - r2l	4,898,432	123	1,126	0.02	743.0 MB
Netflix - 005p	480,189	17,770	264	0.05	**2.6 GB**
Netflix - 05p	480,189	17,770	2,420	0.50	2.6 GB
KDD Cup 1999 - probe	4,898,432	123	41,100	0.83	743.0 MB
Netflix - 1p	480,189	17,770	4,789	1.00	2.6 GB
Dis	3,772	29	58	1.54	270.0 KB
KDD Cup 2009 - appetency	50,000	87,904	890	1.78	1.6 GB
Ann	7,200	21	166	2.30	436.0 KB
Nursery	12,960	8	328	2.53	466.0 KB
Allhyper	3,772	29	102	2.70	270.0 KB
W1a	49,749	300	1,479	2.97	3.4 MB
W8a	64,700	300	1,933	2.98	4.4 MB
Allrep	3,772	29	124	3.29	275.0 KB
Anneal	898	38	40	4.45	80.0 KB
Allbp	2,800	29	133	4.75	200.0 KB
Hypothyroid	3,163	25	151	4.77	281.0 KB
Netflix - 5p	480,189	17,770	23,996	5.00	2.6 GB
Sick	2,800	29	171	6.10	205.0 KB
KDD Cup 2009 - churn	50,000	87,904	3672	7.34	1.6 GB
Abalone	4,177	8	391	9.36	259.0 KB
IJCNN	49,990	22	4,853	9.70	7.6 MB
Netflix - 10p	480,189	17,770	48,317	10.00	2.6 GB
Netflix - 20p	480,189	17,770	76,821	20.00	2.6 GB
Spectheart	267	22	55	20.60	13.0 KB
Hepatitis	155	19	32	20.64	23.0 KB
Wpbc	198	33	47	23.74	61.0 KB
Transfusion	748	4	178	23.80	24.0 KB
A9a	48,842	123	11,687	23.93	3.4 MB
A2a	32,561	123	7,841	24.08	2.3 MB
German	1,000	24	300	30.00	162.0 KB
Real-sim	72,309	20,958	22,238	30.75	88.2 MB
URL Reputation	2,396,130	3,231,961	792,145	33.05	**2.2 GB**
Cod-rna	331,152	8	110,384	33.30	25.4 MB
Pima-India	768	8	268	34.89	41.0 KB
Diabetes	768	8	268	34.90	68.0 KB
HeartDisease	294	13	106	36.00	22.0 KB
BreastCancer	683	10	239	37.99	60.0 KB
Webspam	350,000	254	137,818	39.37	391.0 MB
Australian	690	14	307	44.49	70.0 KB
Sonar	208	60	97	46.63	153.0 KB
Netflix - 47p	480,189	17,770	182,173	47.00	2.6 GB
Rcv1	697,641	47,236	331,690	47.54	1.2 GB
Mushrooms	8,124	112	3,916	48.20	868.0 KB
Splice	1,000	60	483	48.30	699.0 KB
Covtype	581,012	54	283,302	48.76	70.0 MB
News20	19,996	1,355,191	9999	49.99	136.7 MB

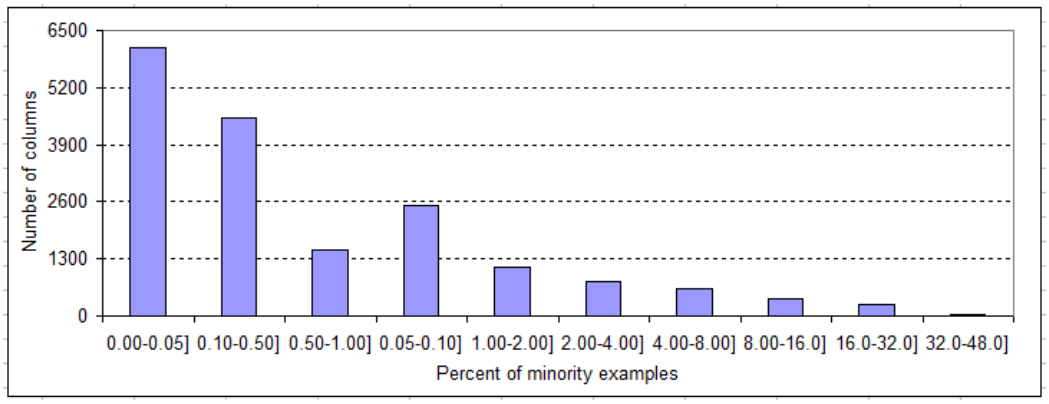


Figure 10.3: Distribution of columns and %minority examples on Netflix data set

10.7.2 Experimental Setting

Protocols

We use 5-fold cross-validation and the t-test with significance level 0.05 for all experiments.

For the cost-sensitive learning as well as the thresholding methods, We implement them using SMO (Platt, 1999a) and Bayesian Networks in WEKA[1].

For experimenting the proposed measure (B42), we compare two classifiers – ℓ_2-regularized logistic regression (ℓ_2-LR) and ℓ_2-loss SVMs (ℓ_2-SVM) – wrt. the AUC, H, and B42. We use the LIBLINEAR[2] software (Fan *et al.*, 2008) with some small modifications to get posterior probability outputs. Here, we experiment on large data sets (as well as small ones), so LIBLINEAR is a better choice over WEKA.

Baselines

We compare the proposed methods with the most popular as well as the state-of-the-art methods such as SMOTE (Chawla *et al.*, 2002), TLINK (Tomek, 1976), MetaCost (Domingos, 1999), the foundation of cost-sensitive learning (CSL) (Elkan, 2001), and cost-sensitive learning by instance weighting (CSW) (Ting, 1998; Witten & Frank, 2005).

For the sampling methods, we denote the scheme as <Sampling method><Percentage>. For example, SM100 and ROS200 denote that the SMOTE and the ROS (random oversampling) with 100% and 200% are used, respectively.

Please note that in this chapter we treat the predicting student problem as a classification problem. So, we have not compared the proposed methods with the Knowledge

[1] www.cs.waikato.ac.nz/ml/weka

[2] www.csie.ntu.edu.tw/∼cjlin/liblinear/

Tracing model (Corbett & Anderson, 1995) as several previous chapters. However, we consider to do this in future work.

Hyper parameters

We use the hyper parameter search procedure as described in Algorithm 10 to determine the hyper parameters for all methods.

10.7.3 Experimental Results

To validate the proposed methods, we have conducted many experiments on both small and large data sets. These data sets come from diversity domains as described in previous sections. Moreover, for checking whether the proposed methods biased on a specific measure as well as for diversity in comparison, we have evaluated them on several evaluation measures, which will be presented in the followings.

S-CSL

For combining sampling technique with cost-sensitive learning (S-CSL), we have implemented four combinations (SMOTE-CSL, TLink-CSL, ROS-CSL, and RUS-CSL) and compared them with three other methods (described in Section 10.7.2).

Figure 10.4 shows the relationship between the cost ratios and the total costs for some typical results. Here, we can see clearly that when the cost ratio increases, the proposed methods reduce the total costs significantly. One reason for this improvement could be because the cost-sensitive learning is a meta-learning method and the internal classifiers (SVM in this case) are still impacted by the class imbalance problem. Thus, it can work better if it is supplied by a re-balanced data set.

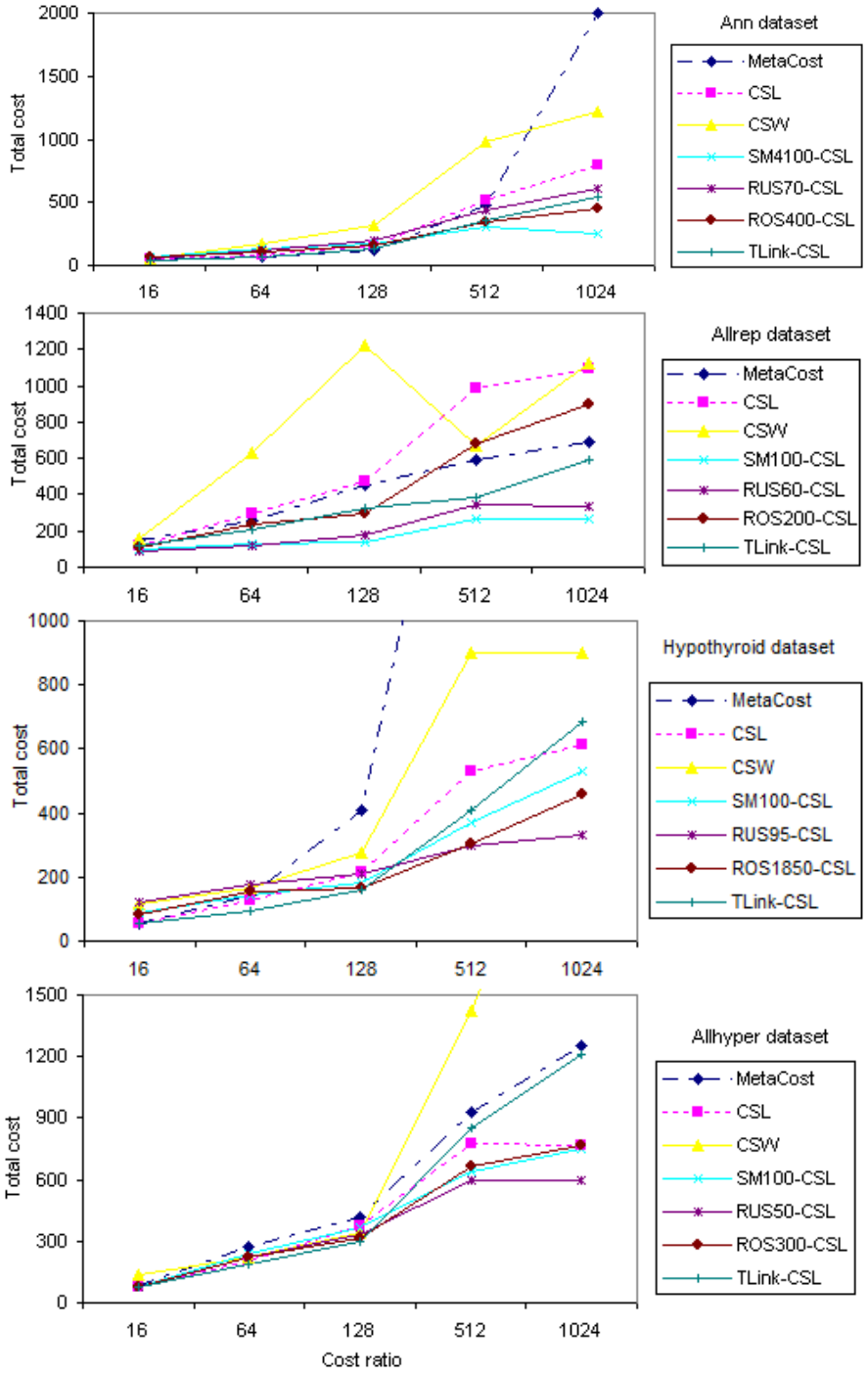


Figure 10.4: Cost ratio and total cost relationship for typical results

Table 10.5: S-CSL: Average costs on other data sets

Data set	SVM	MetaCost	CSL	CSW	***S-CSL***			
					ROS-CSL	RUS-CSL	SMOTE-CSL	TLink-CSL
Abalone (%Sample, Imba. ratio)	6632.00	547.00	486.25	496.90	*486.80*	***460.75***	*486.05*	*469.50*
			(-, 9.68)		(100, 4.85)	(30, 6.79)	(100, 4.85)	(-, 8.97)
Allbp	1771.00	364.50	342.80	373.65	*337.75*	***290.50***	*320.30*	*302.10*
			(-, 20.05)		(300, 5.03)	(20, 16.11)	(500, 3.35)	(-, 19.50)
Allhyper	887.20	373.05	227.00	259.45	*222.30*	*172.60*	*193.20*	***167.90***
			(-, 35.98)		(400, 7.24)	(50, 18.12)	(100, 18.12)	(-, 35.01)
Allrep	1320.00	93.65	112.35	399.25	***84.40***	*84.55*	135.65	111.00
			(-, 29.45)		(300, 7.36)	(80, 5.89)	(2800, 1.01)	(-, 28.79)
Ann	787.00	134.80	131.45	182.05	140.45	***98.10***	*131.05*	*113.40*
			(-, 42.37)		(100, 21.31)	(50, 21.31)	(200, 14.21)	(-, 42.10)
Anneal	154.20	57.40	55.15	55.10	*51.20*	*53.15*	***38.15***	69.30
			(-, 21.45)		(400, 4.28)	(20, 17.15)	(200, 7.14)	(-, 21.25)
Breastcancer	206.20	24.30	25.80	23.60	*14.90*	*14.45*	*24.70*	***13.15***
			(-, 1.90)		(70, 1.12)	(30, 1.33)	(50, 1.27)	(-, 1.85)
Diabetes	1916.80	91.65	89.90	172.45	*85.80*	***86.10***	*86.25*	*90.40*
			(-, 1.86)		(60, 1.16)	(20, 1.49)	(60, 1.16)	(-, 1.52)
Dis	958.80	350.65	422.85	**304.80**	*326.25*	*337.65*	356.05	371.30
			(-, 64.03)		(6000, 1.05)	(50, 32.30)	(500, 10.76)	(-, 63.97)
Heartdisease	598.40	34.10	32.50	195.25	***31.15***	*31.45*	*31.70*	36.30
			(-, 1.77)		(20, 1.51)	(20, 1.44)	(30, 1.38)	(-, 1.28)
Hepatitis	273.60	18.80	19.20	76.55	***18.15***	*18.70*	20.90	19.85
			(-, 3.84)		(50, 2.67)	(30, 2.80)	(100, 1.98)	(-, 3.40)
Hypothyroid	700.00	124.30	140.10	197.20	156.75	*103.45*	***101.50***	*110.55*
			(-, 19.95)		(100, 10.04)	(20, 16.06)	(100, 10.04)	(-, 19.70)
Nursery	1317.8	115.75	108.45	42.75	*42.80*	*37.95*	***26.40***	*35.00*
			(-, 38.51)		(100, 19.28)	(70, 11.57)	(300, 9.64)	(-, 36.56)
Pima-Indian	2017.6	96.3	97.05	128.8	*88.25*	***87.95***	*88.85*	105.00
			(-, 1.87)		(50, 1.24)	(20, 1.49)	(70, 1.10)	(-, 1.50)
Sick	1316.40	254.20	207.00	437.80	*253.35*	*206.65*	***201.20***	256.80
			(-, 15.37)		(200, 5.15)	(20, 12.38)	(100, 7.73)	(-, 15.12)
Spectheart	462.00	34.05	36.75	161.35	40.50	***32.25***	*33.95*	*33.30*
			(-, 3.85)		(200, 1.28)	(50, 1.93)	(50, 2.56)	(-, 2.95)
Transfusion	2840.40	108.6	109.8	105.3	*106.65*	***103.85***	*106.45*	*106.05*
			(-, 3.20)		(50, 2.14)	(10, 2.89)	(50, 2.14)	(-, 2.53)
Wpbc	463.40	33.00	29.90	74.55	*28.40*	***26.35***	*28.05*	*28.55*
			(-, 3.21)		(100, 1.63)	(20, 2.62)	(100, 1.63)	(-, 2.65)
Number of times S-CSL works better than MetaCost/CSL/CSW (out of 18)					*15/13/15*	*18/18/17*	*14/16/15*	*12/12/14*

Note: The lower the cost, the better the model

Table 10.5 compares the results of S-CSL with other methods in term of average costs. For each dataset, when comparing the last four columns (S-CSL) with the other methods, we can see that the average misclassification costs are reduced after re-sampling in most cases. For each row in the table, the **bold number** denotes the best result and the *italic number* describes our combination better than MetaCost. We also report the percentage for the sampling methods, and the imbalance ratio after resampling for each dataset. The combination of RUS with CSL (RUS-CSL) works better than the remaining combinations. In addition, RUS-CSL is always works better than MetaCost, CSL, and CSW (except for the Dis dataset). The last row in the table summarizes the comparison results of each combination with 3 other methods.

Moreover, when observing the imbalance ratio before and after sampling, the results show that not only class imbalance problem, but also noise, borderline examples, and class overlapping may degrade the classifier performance. These problems have also been reported by (Kubat & Matwin, 1997; Prati *et al.*, 2004; Tomek, 1976).

Table 10.6 present the results of the proposed S-CSL method on Algebra and Bridge data. We can observe that a compound of sampling technique (in this case TLink) with cost-sensitive learning can also improve the result on educational data.

Table 10.6: S-CSL: Average costs on Educational Data

Dataset	SVM	MetaCost	CSL	***S-CSL***			
				ROS-CSL	RUS-CSL	SMOTE-CSL	TLink-CSL
Algebra	18660.4	1271.3	1273.7	1271.9	1272.5	1272.5	**1268.0**
	(-, 5.34)			(40, 3.69)	(30, 3.61)	(40, 3.69)	(-, 4.39)
Bridge	27841.2	1424.6	1420.3	1403.4	1403.1	1402.9	**1393.4**
	(-, 3.67)			(40, 2.62)	(30, 2.57)	(40, 2.62)	(-, 3.14)

The numbers in parentheses (#,#) are the sampling percentage and the imbalanced ratio. The lower the cost, the better the model.

O-CSL

The results of locally optimizing the cost ratio for cost-sensitive learning (O-CSL) on educational data sets are presented in the following tables. Table 10.7 shows the average misclassification costs on 5-fold cross-validation. The O-CSL is significantly reduce the costs compared to the other methods.

Table 10.7: O-CSL: Average costs on educational data

Data set	SVM	SMOTE	TLINK	**O-CSL**	CSW	MetaCost
Algebra	1148.0±34.8	1056.2±27.1	986.8±20.8	459.0±13.8	793.2±23.3	809.8±27.6
Bridge	1031.4±32.8	975.0±32.0	940.0±37.8	663.2±33.0	862.2±27.7	875.0±30.7
Average	1089.7	1015.6	963.4	**561.1**	827.7	842.4

The lower the cost, the better the model

Tables 10.8 and 10.9 present the GMean and the true positive rate (TPR) results. Again, the O-CSL also improve to the state-of-the-art SMOTE as well as the others. However, when evaluating by the TPR, the CSW works better on the Bridge data sets.

Table 10.8: O-CSL: GMean on educational data

Data set	SVM	SMOTE	TLINK	**O-CSL**	CSW	MetaCost
Algebra	0.534±0.012	0.620±0.010 ∘	0.667±0.007 ∘	0.782±0.009 ∘	0.774±0.007 ∘	0.754±0.004 ∘
Bridge	0.488±0.022	0.551±0.017 ∘	0.582±0.020 ∘	0.723±0.006 ∘	0.677±0.008 ∘	0.656±0.014 ∘
Average	0.511	0.585	0.625	**0.752**	0.726	0.705

Table 10.9: O-CSL: TPR on educational data

Dataset	SVM	SMOTE	TLINK	**O-CSL**	CSW	MetaCost
Algebra	0.330±0.014	0.420±0.008 ∘	0.474±0.004 ∘	**0.837**±0.008 ∘	0.816±0.013 ∘	0.751±0.010 ∘
Bridge	0.249±0.023	0.327±0.019 ∘	0.370±0.023 ∘	0.816±0.014 ∘	**0.830**±0.015 ∘	0.516±0.024 ∘
Average	0.289	0.374	0.422	**0.827**	0.823	0.634

∘, • statistically significant improvement or degradation

The higher the value, the better the model

Table 10.10: O-CSL: Results of GMean on other data sets

Data set	MetaCost	**O-CSL**	CSW
abalone	0.779±0.020	0.779±0.015	0.784±0.006
allbp	0.865±0.028	**0.870**±0.032	0.823±0.055 •
allhyper	0.893±0.073	**0.895**±0.042	0.841±0.084
allrep	0.874±0.033	**0.886**±0.031	0.789±0.061 •
ann	0.970±0.011	0.949±0.033	0.955±0.041
anneal	0.962±0.057	**0.968**±0.057	0.946±0.055
breastcancer	0.965±0.011	**0.969**±0.016	0.968±0.019
diabetes	0.705±0.046	**0.760**±0.043 ∘	0.746±0.048 ∘
dis	0.738±0.184	**0.739**±0.081	0.641±0.109
heartdisease	0.776±0.049	**0.828**±0.064	0.818±0.067
hepatitis	0.725±0.075	**0.755**±0.061	0.747±0.071
hypothyroid	0.927±0.034	0.899±0.044	0.856±0.038 •
nursery	0.999±0.000	**1.000**±0.000	1.000±0.000
pima	0.697±0.052	**0.747**±0.050	0.737±0.031
sick	0.912±0.033	**0.912**±0.029	0.870±0.054
spectheart	0.730±0.076	**0.772**±0.037	0.732±0.082
transfusion	0.661±0.008	0.678±0.027	0.682±0.021
wpbc	0.689±0.084	0.683±0.056	0.619±0.194
Average	0.826	**0.838**	0.809

∘, • statistically significant improvement or degradation

Table 10.10 compares the results of the O-CSL with the other cost-sensitive learning methods (MetaCost and CSW) using GMean and TPR. The bold numbers present the best results among these methods. On average, the O-CSL also shows improvements.

Thresholding on Bayesian Posterior Probabilities

Table 10.11 presents the detailed results of the F1-Measure and average results of the other metrics (Recall, Precision, and AUC). We report the results of our methods together with 4 other classifiers: Naive Bayes (NB) and Bayesian Networks without optimizing the threshold (BN), SMOTE, and TLINK.

In the first experiment from Table 10.11, we use NB as a baseline. Clearly, we can see that BNOpt, TANOpt, MBOpt, and TLINK easily win this baseline. The EnsBNOpt gives the best average results among the others while BNOpt has more win times (14/6/0). From this experiment, we observed that the NB does not work well. We also optimize the threshold for NB but the results are not better than the other classifiers (we do not report those results here). The reason could be because of its independent assumption, as discussed in the literatures (Friedman *et al.*, 1997; Madden, 2002). Since NB does not work well, in all the remaining experiments, we use the general BN as a base classifier[1] for SMOTE and TLINK which reported in SMOTE and TLINK columns of Table 10.11, respectively.

In the rest experiments, we use general BN as a baseline. We can also see that our methods outperform this baseline. The best classifier in this case is BANOpt (8/12/0). In the third and the fourth experiments, we compare our methods with SMOTE and TLINK, respectively. For example, in third "wins/ties/loses" row, SMOTE is a "base" for comparison, TAN-Opt and BAN-Opt win 8 and tie 12 times (8/12/0) compared to SMOTE. The MBOpt loses once, while the remaining methods work as good as these two sampling methods and even significantly better in certain cases. We also note that the percentage of SMOTE has been optimized for all datasets.

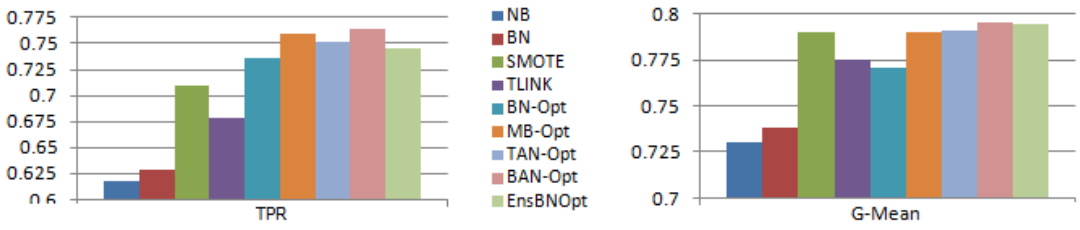


Figure 10.5: Average of TPR and GMean: NB (leftmost), . . . , EnsBNOpt (rightmost)

Figure 10.5 displays the average results of true positive rate (TPR) and GMean on all data sets for NB, BN, SMOTE, TLINK, BNOpt, MBOpt, TANOpt, BANOpt, and EnsBNOpt allocated from the leftmost bar to the rightmost bar, respectively.

[1]Since SMOTE and TLINK are sampling methods, they need a base classifier

When looking at these results, we can also recognize that the TPR - the one which focuses on the interesting class in imbalanced datasets - from our methods significantly outperforms the other methods (NB, BN, TLINK), and slightly better than SMOTE, while the overall models (evaluated by GMean) does not degrade. Those results are the expected ones for all methods which mitigate the class imbalance problem.

In this work, we just optimize the results for F1-Measure, but for referencing, we also report the average results on 20 datasets of the Recall, Precision, and AUC in the last 3 rows of Table 10.11. The BAN-Opt again shows the best performance among the others for Recall, while EnsBNOpt shows the best Precision and AUC on average.

Table 10.12 presents the experimental results of learning optimal threshold on Bayesian posterior probabilities on educational data.

Table 10.11: Details of F1-Measure and Average of other Metrics

Dataset	NBayes	BayesNet	SMOTE	TLINK	Learn-BNOpt				Learn-EnsBNOpt	
					BNOpt	MBOpt	TANOpt	BANOpt		
abalone	.361±.014	.380±.013 ○	.379±.018 ○	.380±.014 ○	.407±.024 ○	.417±.018 ○	.432±.031 ○	.421±.022 ○	.429±.026	○
allbp	.522±.077	.603±.065 ○	.598±.070 ○	.589±.068 ○	.606±.036 ○	.636±.074 ○	.569±.052	.608±.028 ○	.608±.047	
allhyper	.490±.071	.553±.105 ○	.495±.098	.554±.091 ○	.577±.120 ○	.708±.076 ○	.661±.108 ○	.707±.109 ○	.670±.150	○
allrep	.407±.066	.664±.075 ○	.632±.076 ○	.652±.049 ○	.657±.087 ○	.894±.043 ○	.826±.063 ○	.862±.038 ○	.880±.046	○
ann	.801±.053	.907±.037 ○	.901±.035 ○	.887±.056 ○	.922±.043 ○	.932±.031 ○	.938±.021 ○	.941±.027 ○	.935±.031	○
anneal	.583±.135	.920±.089 ○	.874±.078 ○	.920±.089 ○	.920±.089 ○	.920±.089 ○	.920±.089 ○	.920±.089 ○	.920±.089	○
breastcancer	.944±.011	.960±.014	.959±.014	.962±.011 ○	.962±.011 ○	.943±.015	.955±.014	.945±.018	.949±.010	
diabetes	.645±.089	.646±.077	.651±.054	.662±.071	.646±.067	.644±.055	.645±.056	.645±.056	.652±.052	
dis	.276±.031	.521±.085 ○	.430±.058 ○	.477±.075 ○	.514±.098 ○	.490±.090 ○	.496±.083 ○	.504±.091 ○	.484±.094	○
haberman	.303±.174	.155±.214	.483±.059	.465±.154	.404±.023	.404±.023	.404±.023	.404±.023	.404±.023	
heartdisease	.795±.062	.743±.083	.753±.096	.750±.075	.749±.088	.754±.077	.738±.084	.754±.077	.739±.086	
hepatitis	.641±.135	.479±.224	.644±.150	.603±.201	.521±.269	.429±.274	.555±.196	.480±.315	.564±.164	
hypothyroid	.778±.035	.871±.032 ○	.841±.034 ○	.867±.042 ○	.865±.024 ○	.844±.056 ○	.860±.049 ○	.857±.024 ○	.862±.026	○
ijcnn	.304±.019	.410±.025 ○	.515±.015 ○	.415±.023 ○	.525±.008 ○	.623±.017 ○	.573±.030 ○	.609±.021 ○	.614±.029	○
nursery	.380±.035	.387±.035	.680±.039 ○	.548±.021 ○	.768±.020 ○	.859±.015 ○	.859±.015 ○	.859±.015 ○	.852±.024	○
pima	.634±.089	.636±.073	.635±.066	.647±.081	.628±.042	.655±.044	.643±.058	.647±.054	.650±.047	
sick	.554±.071	.757±.052 ○	.703±.093 ○	.746±.047 ○	.780±.074 ○	.803±.063 ○	.777±.065 ○	.794±.059 ○	.804±.065	○
tictactoe	.501±.064	.501±.064	.537±.056	.502±.053	.599±.021 ○	.680±.034 ○	.658±.035 ○	.709±.041 ○	.713±.040	○
transfusion	.282±.033	.488±.027 ○	.491±.026 ○	.486±.022 ○	.483±.032 ○	.488±.021 ○	.488±.021 ○	.488±.021 ○	.488±.021	○
wine-red	.493±.075	.484±.047	.491±.025	.499±.054	.490±.044	.516±.069	.488±.028	.503±.043	.509±.038	
F1-Measure AVG	.535	.603	.635	.630	.651	.682	.674	.683	**.686**	
wins/ties/loses	base	11/9/0	11/9/0	13/7/0	**14/6/0**	13/7/0	12/8/0	13/7/0	12/8/0	
wins/ties/loses	0/9/11	base	3/15/2	1/19/0	4/16/0	7/12/1	7/13/0	**8/12/0**	7/13/0	
wins/ties/loses	0/9/11	2/15/3	base	2/15/3	4/16/0	8/11/1	**8/12/0**	**8/12/0**	7/13/0	
wins/ties/loses	0/7/13	0/19/1	3/15/2	base	5/15/0	9/10/1	6/14/0	**8/12/0**	**8/12/0**	
Recall average	.617	.628	.709	.679	.736	.759	.751	**.764**	.745	
Precision	.559	.635	.601	.631	.611	.642	.636	.639	**.660**	
AUC average	.879	.885	.888	.889	.885	.894	.893	.894	**.895**	

○, ● statistically significant improvement or degradation

Table 10.12: Thresholding on Bayesian Posterior Probabilities - Educational data

A. GMean

Data set	BayesNet	SMOTE	TLINK	BNOpt	MBOpt	TANOpt	BANOpt	EnsBNOpt
Algebra	0.569±0.030	0.668±0.016 ○	0.672±0.024 ○	0.737±0.047 ○	0.754±0.043 ○	0.754±0.043 ○	0.754±0.043 ○	0.754±0.043 ○
Bridge	0.444±0.070	0.583±0.018 ○	0.592±0.023 ○	0.700±0.019 ○	0.704±0.014 ○	0.704±0.014 ○	0.704±0.014 ○	0.700±0.019 ○
Average	0.507	0.626	0.632	0.718	0.729	0.729	0.729	0.727

-

B. F-Measure

Data set	BayesNet	SMOTE	TLINK	BNOpt	MBOpt	TANOpt	BANOpt	EnsBNOpt
Algebra	0.438±0.035	0.525±0.012 ○	0.525±0.019 ○	0.549±0.018 ○	0.546±0.021 ○	0.546±0.021 ○	0.546±0.021 ○	0.546±0.021 ○
Bridge	0.306±0.073	0.442±0.022 ○	0.450±0.026 ○	0.512±0.017 ○	0.514±0.017 ○	0.514±0.017 ○	0.514±0.017 ○	0.514±0.017 ○
Average	0.372	0.483	0.487	0.531	0.530	0.530	0.530	0.530

-

C. AUC

Data set	BayesNet	SMOTE	TLINK	BNOpt	MBOpt	TANOpt	BANOpt	EnsBNOpt
Algebra	0.862±0.005	0.863±0.005	0.862±0.006	0.862±0.005	0.862±0.005	0.862±0.005	0.862±0.005	0.863±0.005
Bridge	0.785±0.009	0.786±0.011	0.786±0.011	0.785±0.009	0.786±0.010	0.786±0.010	0.786±0.010	0.787±0.009
Average	0.824	0.824	0.824	0.824	0.824	0.824	0.824	0.825

○, ● statistically significant improvement or degradation

Thresholding on Sampling Data

Table 10.13 presents the detailed results of learning optimal threshold on sampling data, evaluated by using F1-Measure and other metrics. Here, we use the SMOTE as a sampling method in the proposed Learn-S-BNOpt. We report the results together with 4 other classifiers: Naive Bayes, general BN without optimizing the threshold, SMOTE, and TLINK.

Clearly, the proposed method MBOpt, TANOpt, and BANOpt outperform the state-of-the-art SMOTE 7 significant results out of 16 datasets (7/9/0). The MBOpt gives the best average result of F1-Measure among the others.

Table 10.14 shows the results of using TLINK, ROS, and RUS as a sampling method in the Learn-S-BNOpt, respectively. The MBOpt and BANOpt also outperform the other methods. From these results, we recognize that the BANOpt can perform better if we remove "noise" and "borderline" examples by using TLINK or remove randomly by RUS, while the MBOpt requires more artificial data generated by SMOTE or ROS to get better performance.

Tables 10.15 and 10.16 present the results of learning optimal threshold on sampling data, for Algebra and Bridge data sets.

Table 10.13: Detailed results of F1-Measure and average of other Metrics

Data set	NaiveBayes	BayesNet	SMOTE	TLINK	Learn-R-Opt-BNs (R is SMOTE)				
					BN-Opt	MB-Opt	TAN-Opt	BAN-Opt	BNCost
abalone	.361± .014	.370±.013 ∘	.379±.018 ∘	.380±.014 ∘	.410±.020 ∘	**.416**±.017 ∘	.410±.018 ∘	.407±.021 ∘	.380±.013 ∘
allbp	.522± .077	.559±.069 ∘	.598±.070 ∘	.589±.068 ∘	.563±.080	.579±.049	.586±.075	.559±.055	**.603**±.065 ∘
allhyper	.490± .071	.519±.086 ∘	.495±.098	.554±.091 ∘	.548±.108	**.691**±.066 ∘	.632±.112 ∘	.663±.058 ∘	.553±.105 ∘
allrep	.407± .066	.519±.067 ∘	.632±.076 ∘	.652±.049 ∘	.665±.060 ∘	**.829**±.056 ∘	.753±.030 ∘	.757±.049 ∘	.664±.075 ∘
ann	.801± .053	.852±.043 ∘	.901±.035 ∘	.887±.056 ∘	.921±.023 ∘	.931±.025 ∘	**.933**±.029 ∘	.928±.026 ∘	.907±.037 ∘
anneal	.583± .135	.699±.115 ∘	.874±.078 ∘	**.920**±.089 ∘	.833±.108 ∘	.908±.079 ∘	.881±.074 ∘	.889±.109 ∘	**.920**±.089 ∘
breastcancer	.944± .011	.952±.010	.959±.014	.962±.011 ∘	**.964**±.011 ∘	.956±.018	.948±.018	.949±.012	.960±.014
diabetes	.645± .089	.646±.082	.651±.054	.662±.071	**.668**±.025	.667±.046	.667±.054	.670±.044	.646±.077
dis	.276± .031	.362±.018 ∘	.397±.075	.477±.075 ∘	.398±.057 ∘	.300±.120	.418±.095 ∘	.325±.107	**.521**±.085 ∘
heartdisease	**.795**± .062	.770±.062	.753±.096	.750±.075	.742±.096	.763±.078	.746±.090	.749±.089	.743±.083
hypothyroid	.778± .035	.826±.027 ∘	.841±.034 ∘	.867±.042 ∘	.832±.030 ∘	.837±.062	.856±.046 ∘	.850±.030 ∘	**.871**±.032 ∘
ijcnn	.304± .019	.359±.020 ∘	.515±.015 ∘	.415±.023 ∘	.538±.005 ∘	**.653**±.006 ∘	.579±.015 ∘	.635±.015 ∘	.410±.025 ∘
nursery	.380± .035	.384±.034	.680±.039 ∘	.548±.021 ∘	.738±.022 ∘	**.897**±.026 ∘	**.897**±.026 ∘	**.897**±.026 ∘	.387±.035
pima	.634± .089	.635±.078	.635±.066	.647±.081	**.653**±.038	.645±.055	.639±.054	.642±.047	.636±.073
sick	.554± .071	.642±.063 ∘	.703±.093 ∘	.746±.047 ∘	.762±.085 ∘	**.798**±.064 ∘	.776±.066 ∘	.779±.062 ∘	.757±.052 ∘
transfusion	.282± .033	.404±.014 ∘	.491±.026 ∘	.486±.022 ∘	.450±.026 ∘	**.492**±.014 ∘	**.492**±.014 ∘	**.492**±.014 ∘	.488±.027 ∘
F1 Average	.547	.594	.657	.659	.668	**.710**	.701	.700	.653
AUC Average	.905	.909	.915	.917	.915	**.923**	.920	.921	.917
GMean Average	.754	.773	.813	.802	.816	**.831**	**.831**	.829	.789
Recall Average	.649	.668	.728	.717	.778	.784	**.787**	.786	.688

Paired t-tests with significance level .05; ∘, • statistically significant improvement or degradation

Table 10.14: Average results: Thresholding on Sampling Data

A. TLINK is used as a sampling method in Learn-S-BNOpt

Dataset	NaiveBayes	BayesNet	SMOTE	TLINK	Learn-S-BNOpt (S is TLINK)				
					BNOpt	MBOpt	TANOpt	BANOpt	BNCost
F1 Average	0.547	0.594	0.657	0.659	0.683	**0.728**	0.711	**0.728**	0.653
AUC Average	0.905	0.909	0.915	0.917	0.917	**0.922**	0.921	0.921	0.917
GMean Average	0.754	0.773	0.813	0.802	0.820	0.840	0.825	**0.844**	0.789
Recall Average	0.649	0.668	0.728	0.717	0.779	0.792	0.775	**0.798**	0.688

B. ROS is used as a sampling method in Learn-S-BNOpt

Dataset	NaiveBayes	BayesNet	SMOTE	TLINK	Learn-S-BNOpt (S is ROS)				
					BNOpt	MBOpt	TANOpt	BANOpt	BNCost
F1 Average	0.547	0.594	0.657	0.659	0.679	**0.719**	0.705	0.715	0.653
AUC Average	0.905	0.909	0.915	0.917	0.918	**0.925**	0.923	0.923	0.917
GMean Average	0.754	0.773	0.813	0.802	0.832	0.839	0.841	**0.846**	0.789

C. RUS is used as a sampling method in Learn-S-BNOpt

Data set	NaiveBayes	BayesNet	SMOTE	TLINK	Learn-S-BNOpt (S is RUS)				
					BNOpt	MBOpt	TANOpt	BANOpt	BNCost
F1 Average	0.547	0.594	0.657	0.659	0.675	**0.720**	0.708	0.716	0.653
AUC Average	0.905	0.909	0.915	0.917	0.917	**0.922**	**0.922**	**0.922**	0.917
GMean Average	0.754	0.773	0.813	0.802	0.825	0.847	0.844	**0.852**	0.789

Table 10.15: Thresholding on sampling data (using SMOTE)

A. G-Mean

Data set	BayesNet	SMOTE	TLINK	SMOTE-BNOpt	SMOTE-MBOpt	SMOTE-TANOpt	SMOTE-BANOpt	Cost4-BN
Algebra	0.573±0.016	0.668±0.016 ○	0.672±0.024 ○	0.779±0.007 ○	0.767±0.026 ○	0.767±0.026 ○	0.767±0.026 ○	0.778±0.005 ○
Bridge	0.487±0.035	0.583±0.018 ○	0.592±0.023 ○	0.708±0.009 ○	0.713±0.013 ○	0.713±0.013 ○	0.713±0.013 ○	0.706±0.010 ○
Average	0.530	0.626	0.632	0.744	0.740	0.740	0.740	0.742

-

B. AUC

Data set	BayesNet	SMOTE	TLINK	SMOTE-BNOpt	SMOTE-MBOpt	SMOTE-TANOpt	BANOpt	Cost4-BN
Algebra	0.854±0.006	0.863±0.005 ○	0.862±0.006 ○	0.863±0.005 ○	0.864±0.005 ○	0.864±0.005 ○	0.864±0.005 ○	0.862±0.005 ○
Bridge	0.773±0.010	0.786±0.011 ○	0.786±0.011 ○	0.786±0.011 ○	0.786±0.013 ○	0.786±0.013 ○	0.786±0.013 ○	0.785±0.009 ○
Average	0.813	0.824	0.824	0.824	0.825	0.825	0.825	0.824

-

C. FMeasure

Dataset	BayesNet	SMOTE	TLINK	SMOTE-BNOpt	SMOTE-MBOpt	SMOTE-TANOpt	SMOTE-BANOpt	Cost4-BN
Algebra	0.437±0.021	0.525±0.012 ○	0.525±0.019 ○	0.550±0.017 ○	0.553±0.013 ○	0.553±0.013 ○	0.553±0.013 ○	0.543±0.021 ○
Bridge	0.350±0.037	0.442±0.022 ○	0.450±0.026 ○	0.507±0.011 ○	0.511±0.015 ○	0.511±0.015 ○	0.511±0.015 ○	0.505±0.010 ○
Average	0.393	0.483	0.487	0.528	0.532	0.532	0.532	0.524

○, • statistically significant improvement or degradation

Table 10.16: Thresholding on Sampling Data (using TLINK)

A. G-Mean

Dataset	BayesNet	SMOTE	TLINK	TLINK-BNOpt	TLINK-MBOpt	TLINK-TANOpt	TLINK-BANOpt	Cost4-BN
Algebra	0.573±0.016	0.668±0.016 ∘	0.672±0.024 ∘	0.759±0.026 ∘	0.759±0.026 ∘	0.759±0.026 ∘	0.759±0.026 ∘	0.778±0.005 ∘
Bridge	0.487±0.035	0.583±0.018 ∘	0.592±0.023 ∘	0.703±0.018 ∘	0.711±0.016 ∘	0.711±0.016 ∘	0.711±0.016 ∘	0.706±0.010 ∘
Average	0.530	0.626	0.632	0.731	0.735	0.735	0.735	0.742

-

B. AUC

Data set	BayesNet	SMOTE	TLINK	TLINK-BNOpt	TLINK-MBOpt	TLINK-TANOpt	TLINK-BANOpt	Cost4-BN
Algebra	0.854±0.006	0.863±0.005 ∘	0.862±0.006 ∘	0.862±0.006 ∘	0.861±0.006 ∘	0.861±0.006 ∘	0.861±0.006 ∘	0.862±0.005 ∘
Bridge	0.773±0.010	0.786±0.011 ∘	0.786±0.011 ∘	0.786±0.011 ∘	0.791±0.012 ∘	0.791±0.012 ∘	0.791±0.012 ∘	0.785±0.009 ∘
Average	0.813	0.824	0.824	0.824	0.826	0.826	0.826	0.824

-

C. FMeasure

Data set	BayesNet	SMOTE	TLINK	TLINK-BNOpt	TLINK-MBOpt	TLINK-TANOpt	TLINK-BANOpt	Cost4-BN
algebra	0.437±0.021	0.525±0.012 ∘	0.525±0.019 ∘	0.541±0.029 ∘	0.541±0.029 ∘	0.541±0.029 ∘	0.541±0.029 ∘	0.543±0.021 ∘
bridge	0.350±0.037	0.442±0.022 ∘	0.450±0.026 ∘	0.513±0.017 ∘	0.519±0.017 ∘	0.519±0.017 ∘	0.519±0.017 ∘	0.505±0.010 ∘
Average	0.393	0.483	0.487	0.527	0.530	0.530	0.530	0.524

∘, • statistically significant improvement or degradation

A new evaluation measure (B42)

Table 10.17 presents the detailed results of three metrics: the proposed B42, the AUC, and the H measure.

The AUC evaluates ℓ_2-LR outperforming ℓ_2-SVM (at least equal) on 3 groups, while B42 shows that when the imbalance ratio increases, ℓ_2-LR shifts from win (3/9/0) to lose (1/8/3) results, as illustrated in Table 10.18. For example, 1/8/3 means that the ℓ_2-LR wins one time, ties eight times, and loses three times, compared to the ℓ_2-SVM.

Tables 10.19 and 10.20 summarize the agreed/disagreed results of B42 vs. AUC and B42 vs. H on 36 data sets when comparing ℓ_2-LR with ℓ_2-SVM (base). The bold number in the diagonal (e.g. 10 and 7) means that B42 evaluates ℓ_2-LR significantly outperforming/degrading ℓ_2-SVM 10 times, but that AUC disagrees on those results, while the reverse is 7 times the case. These agreed/disagreed results could be because the B42 places more weight on the minority examples, thus, it has more statistically significant improvements or degradations compared to the AUC and the H. However, a deeper analysis needs to be done here. The results are presented in the next paragraph.

Let us analyze more details for the specific data set "nf-05p" in Figure 10.6, which displays an example of cost weight distribution implicitly used in the AUC (for "nf-05p") and explicitly used in B42 and H.

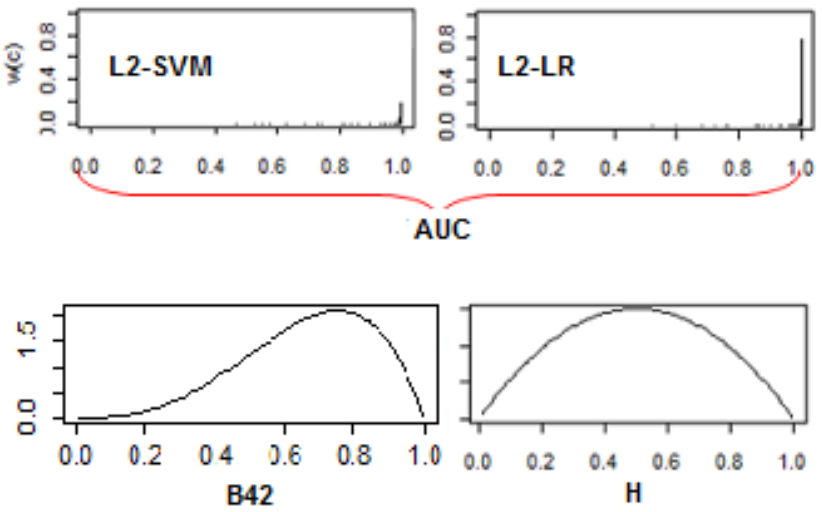


Figure 10.6: Cost weight distribution of the AUC (on nf-05p data set), B42, and H

Clearly, the AUC places *different cost weight distributions for ℓ_2-LR (higher at 1.0) and ℓ_2-SVM on the same "nf-05p" data set.* This means that the AUC uses different metrics to evaluate different classifiers Hand (2009), while B42 and H use the same distribution for all data sets and classifiers. This is the reason why the result of ℓ_2-LR significantly outperforms ℓ_2-SVM regarding the AUC while it only ties regarding B42.

Table 10.17: Results of ℓ_2-SVM (base) and ℓ_2-LR on B42, AUC, and H

	B42			AUC			H		
Data set	ℓ_2-SVM	ℓ_2-LR		ℓ_2-SVM	ℓ_2-LR		ℓ_2-SVM	ℓ_2-LR	
r2l(♢)	.413±.371	.519±.356		.963±.082	.980±.044		.345±.290	.444±.278	
nf-005p (♣)	.006±.002	.005±.003		.523±.033	.617±.038	∘	.002±.001	.003±.002	
nf-05p (♡)	.005±.001	.022±.005		.628±.008	.767±.013	∘	.002±.001	.010±.001	
probe (♢)	.358±.364	.543±.283		.726±.119	.818±.122		.324±.270	.467±.196	
nf-1p(♣)	.010±.001	.039±.002		.670±.007	.784±.004	∘	.005±.001	.019±.001	
appetency (♡)	.012±.003	.026±.007		.735±.020	.775±.014	∘	.007±.003	.013±.005	
ann	.615±.093	.659±.068		.929±.025	.984±.010	∘	.536±.105	.591±.057	
allhyper	.459±.125	.328±.105	•	.862±.084	.886±.030		.254±.226	.227±.116	
w1a	.169±.183	.214±.106	∘	.810±.108	.853±.034		.108±.124	.160±.133	∘
allrep	.441±.064	.385±.058	•	.970±.006	.967±.008		.343±.072	.264±.042	•
anneal	.635±.155	.411±.116	•	.957±.028	.911±.042		.573±.181	.392±.297	
allbp	.374±.169	.324±.082		.886±.135	.859±.112		.280±.132	.207±.081	
hypothyroid	.134±.179	.343±.139	∘	.834±.103	.843±.056		.292±.208	.266±.143	
nf-5p (♣)	.112±.005	.126±.005		.766±.036	.804±.004		.061±.003	.068±.003	
sick	.625±.071	.596±.065	•	.929±.035	.941±.023		.535±.080	.517±.070	
churn (♡)	.011±.003	.023±.005		.605±.017	.648±.018	∘	.005±.002	.011±.002	
abalone	.206±.054	.205±.054		.847±.024	.845±.024		.125±.043	.122±.041	
ijcnn	.313±.158	.300±.150		.861±.059	.858±.058		.227±.144	.214±.135	
nf-10p (♣)	.194±.008	.226±.010	∘	.756±.005	.817±.006	∘	.118±.006	.137±.007	
nf-20p (♣)	.223±.004	.237±.003		.752±.003	.772±.003		.149±.003	.157±.003	
hepatitis	.422±.195	.484±.147		.645±.368	.736±.257		.344±.234	.417±.341	
transfusion	.060±.077	.399±.253	∘	.562±.177	.761±.178		.132±.142	.372±.255	
a9a	.266±.033	.270±.009		.792±.006	.794±.005		.176±.006	.178±.007	
a2a	.293±.011	.318±.011	∘	.792±.009	.793±.005		.178±.013	.180±.007	
real-sim	.474±.278	.768±.200	∘	.812±.218	.959±.057		.455±.263	.735±.231	∘
url	.075±.021	.095±.034		.546±.025	.565±.045		.072±.020	.086±.032	
cod-rna	.166±.115	.108±.076		.586±.226	.640±.094		.155±.106	.079±.056	
pima	.216±.144	.169±.085		.587±.197	.621±.037		.186±.124	.158±.077	
diabetes	.396±.052	.398±.049		.671±.055	.695±.052		.144±.059	.165±.072	
heartdisease	.024±.034	.108±.090	∘	.317±.177	.550±.105	∘	.023±.033	.092±.068	
breastcancer	.087±.096	.138±.144		.404±.158	.488±.176	∘	.099±.112	.150±.154	
nf-47p (♣)	.006±.001	.007±.000		.463±.003	.462±.002		.007±.001	.008±.001	
rcv1	.006±.004	.086±.022	∘	.533±.015	.559±.025		.006±.004	.063±.017	∘
splice	.106±.024	.111±.027		.584±.030	.589±.031		.084±.020	.086±.021	
covtype	.087±.101	.103±.112		.533±.106	.543±.108		.072±.084	.084±.090	
news20	.099±.074	.088±.046		.490±.251	.486±.241		.101±.169	.091±.146	
Average	.225	.256		.703	.749		.181	.202	

(♢): KDD Cup 1999 data set; (♡): KDD Cup 2009 data set; (♣): Netflix data set.
∘, • statistically significant improvement or degradation (level=0.05).

Table 10.18: Win/tie/lose results aggregated from Table 10.17 to 3 groups; ℓ_2-SVM (base) vs. ℓ_2-LR

Group (12 data sets / group)	%Minority	B42	AUC	H
Group 1 (highly imbalanced)	0.02 - 5	**1/8/3**	5/7/0	1/10/1
Group 2	5 - 30	4/7/1	2/10/0	0/12/0
Group 3 (nearly balanced)	30 - 49	**3/9/0**	2/10/0	2/10/0

Table 10.19: The B42 disagrees with the AUC 17 times out of 36 data sets

	Significant difference	Not significant difference
Significant difference	2	**7**
Not significant difference	**10**	17

Table 10.20: The B42 disagrees with the H 8 times out of 36 data sets

	Significant difference	Not significant difference
Significant difference	4	**0**
Not significant difference	**8**	24

The same situation happens with other data sets, e.g., "nf-005p", "nf-1p" and "ann".

Furthermore, Figure 10.7 shows four typical results of the AUC, the true positive rate, and the B42. We can see that the AUC evaluates the ℓ_2-LR outperforming the ℓ_2-SVM, however, the true positive rate and the B42 show the reversed results.

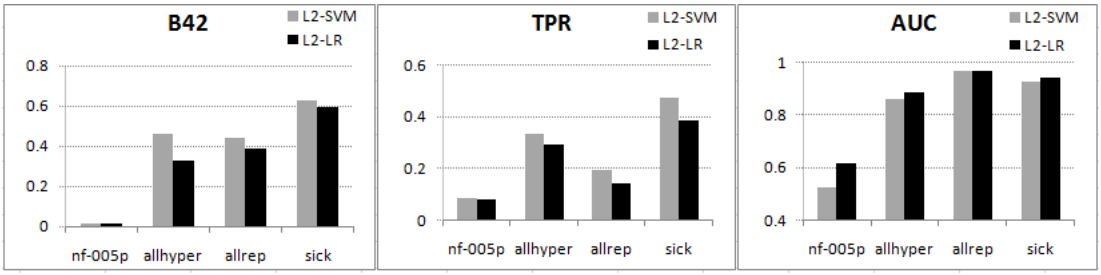


Figure 10.7: Typical results of the AUC, the True positive rate, and the B42

The B42 is consistent with the true positive rate while the AUC is not. Thus, if we would like to take the minority class into account then the B42 is a better choice.

In addition, the empirical results also show that B42 is not only suitable for evaluating on imbalanced data but also for evaluating on balanced data sets (in group 3 in Table 10.17, its results are also consistent with other metrics, e.g the H measure).

10.8 Conclusion

In this chapter we introduce several methods for dealing with imbalanced data. These methods belong to the combination approach such as combining cost-sensitive learning or thresholding with sampling methods. The proposed methods can be used to deal with the class imbalance problem not only in educational domain but also in other areas (e.g., network intrusion detection, fraud detection, etc) where the data sets are skewed in their class distributions. Furthermore, we have also proposed a new evaluation measure for evaluating the classifiers when learning from imbalanced data.

To validate the proposed methods, many experiments have been conducted. These experimental results show that, in most of the cases, the proposed methods perform nicely compared to the other state-of-the-art methods in this domain.

Chapter 11

Conclusion

Contents

11.1 Summary 147
11.2 Discussion 151
11.3 Future Direction 154

11.1 Summary

Predicting student performance is an important task in Student Modeling, where we can give the students early feedback to help them learning better. A good and reliable model which accurately predicts student performance may replace the current standardized tests, thus, reducing pressure on teaching and learning for examinations as well as saving much time and effort for both the teachers and the students (Cen *et al.*, 2007; Feng *et al.*, 2009).

In this work, we have contributed many results to several communities, e.g., to the Intelligent Tutoring Systems, Educational Data Mining, and Recommender Systems. The main contributions are summarized in the following.

First, we have formulated the problem of predicting student performance. We have shown how to map the student performance prediction problem to a rating prediction task in recommender systems, to a forecasting problem, and to regression/classification problems.

Second, we have proposed using latent factor models (e.g. matrix factorization and biased matrix factorization) for student modeling. These models could implicitly take into account the student and task latent factors (e.g. slip and guess), the "student effect/bias" (e.g., how good/clever a student is) and the "task effect/bias" (e.g., how difficult/easy a task is). Moreover, since the students who have similar performances on the past problems may also have similar performances on the future problems,

we propose taking into account the correlations between students and tasks by using user/item k-nearest neighbors collaborative filtering.

Third, in predicting student performance, each student performs several tasks, and each task requires one or several skills, while the students are also required to master the skills that they have learned. We have proposed to exploit such multiple relationships by using (weighted) multi-relational matrix factorization approach.

Fourth, as an expected result from knowledge acquisition, the student performance (or student knowledge) cumulates and improves over time, thus, there is a "trend line" in the student performance. Furthermore, there is a natural fact that if student A is clever (having higher performance) than student B then we should not use B's performance to predict A's. We should either use the performances of the students who are "similar to the student A" as in the collaborative filtering approach or use the proposed personalized forecasting methods which utilize the information of individual student for forecasting his/her own performance.

Fifth, the student's knowledge is assumed to be changing over time and we realized that the second time the student does his/her exercises, his/her performance gets better on average. Thus, temporal/sequential information obviously has an important effect on the student performance. Taking into account these issues we have proposed tensor factorization models to incorporate the forecasting techniques into the latent factor models for modeling the sequential/temporal effects.

Sixth, we have proposed the student's task recommendation which based on student performance. This approach can tackle the following issue in the literature: "the preferred learning activities of students might pedagogically not be the most adequate" since we can recommend the tasks to the students using "student performance" instead of "student preference". With student performance, we can recommend the suitable tasks to the students by filtering out the tasks that are too easy (high predicted performance) or too hard (low predicted performance), or both, depending on the system goal. Furthermore, contradictory from the recommender systems where each user is assumed to rate for an item once, in e-learning environment each student may perform a task several times. We have suggested using context-aware factorization models to utilize the multiple interactions between students and tasks.

Seventh, we have discovered a characteristic in student performance data, which is the class imbalance problem (e.g., the number of correct solutions are higher than the number of incorrect solutions). To alleviate this problem, we have introduced several methods which can be used not only for educational data but also for other applications in machine learning (e.g., fraud detection, etc). We also propose a new evaluation measure for evaluating the classifiers when learning from imbalanced data environments.

Finally, we have evaluated the proposed methods using both small and large published data sets. We have compared them with the other state-of-the-art methods in student modeling, in recommender systems, as well as in imbalanced learning. We empirically show that, in most of the cases, the proposed methods can improve the

prediction results compared to the others.

Now, it's time to put all the methods together for overall comparisons. Table 11.1 summarizes the RMSE comparisons using three data sets for all proposed methods and the other methods. In these results, we have used I_7 as an item (I_7 is the solving-step on Algebra/Bridge and is the problem on ASSISTments, please refer back to Table 3.3 for details).

Please note that, all these methods are developed for the *regression problem* while the methods for dealing with class imbalance in Chapter 10 are proposed for the *classification problem*. We have used different measures (different loss/score functions), therefore, we will not compare imbalanced learning methods here.

Table 11.1: Overall comparison: Using solving-step as an item (the methods are ranked by using RMSE)

Algebra Data Set		Bridge Data Set		ASSISTments Data Set	
Method	RMSE	Method	RMSE	Method	RMSE
TFF	0.3070	B-PDMF	0.2852	TFF	0.4018
TFW	0.3072	TFF	0.2870	TFMAF	0.4027
TFMAF	0.3075	B-PDEF	0.2873	TFW	0.4037
B-PDMF	0.3076	TFMAF	0.2881	TFA	0.4039
TFA	0.3078	WMRMF	0.2883	BPDMF	0.4114
B-PDEF	0.3085	TFW	0.2896	BPSEF	0.4133
B-PSEF	0.3094	B-PSEF	0.2896	BPDEF	0.4149
WMRMF	0.3099	TFA	0.2900	WMRMF	0.4251
BMF	0.3105	BMF	0.2903	Student-kNN-Cosine	0.4269
MRMF	0.3117	MRMF	0.2925	Student-kNN-Pearson	0.4278
MF	0.3129	MF	0.2935	BMF	0.4289
Student-kNN-Cosine	0.3130	Logistic Regression	0.2960	MRMF	0.4344
Student-kNN-Pearson	0.3131	Biased-Student-Task	0.3006	MF	0.4363
Logistic Regression	0.3132	Student-kNN-Cosine	0.3017	Biased Student-Task	0.4391
CFAR	0.3196	CFAR	0.3054	Logistic Regression	0.4439
Biased Student-Task	0.3197	Student kNN-Pearson	0.3072	Student Average	0.4448
Student Average	0.3287	Student Average	0.3114	CFAR	0.4566
Global Average	0.3328	Global Average	0.3186	Global Average	0.4705
Task-kNN-Cosine	n/a	Task kNN-Cosine	n/a	Task-kNN-Cosine	n/a

For the Task-kNN methods, the number of tasks are quite large (refer back to Table 3.3 for these numbers) and we could not manage the computer memory for calculating the task-similarity matrix, thus, we have not reported their results here.

On Algebra and ASSISTments data sets, seven results on the top of Table 11.1 belong to the methods which take into account the temporal/sequential effect, while on Bridge data set, seven-top results also belong to the tensor factorization and personalized forecasting approaches (except the WMRMF). This means that the sequen-

tial/temporal effect is really an important issue in predicting student performance where the student knowledge changes and improves over time.

Surprisingly, on the Bridge data, the very simple and very fast method - B-PDMF (Biased Personalized Discounted-Mean Forecaster) outperforms the other models including the TFF. One of the reasons could be that on the Bridge, which is the largest data set, the TFF runs rather slow (e.g., ≈15 hours) so we have not determined good hyper parameters for it (we have just done a raw search as described in Section 5.5.2). Another reason could be because the personalized forecasting B-PDMF is not affected by the new-item problem while the TFF is (for instance, we just simply returned the global average for these new tasks). Moreover, the TFF has to face with the sparsity problem (99.63% sparsity on the Bridge) while the forecasting B-PDMF has not.

Taking into account multiple relationships between students and tasks by using multi-relational approach, e.g. the WMRMF, provides more benefit than using the single relational model (e.g., the MF).

The results are not clearly different between the logistic regression and the kNN collaborative filtering methods (Student-kNN and Task-kNN) since on the Algebra and ASSISTments data set, the kNNs work better than the logistic regression while the cases are reversed on the Bridge data set. Overall, when using solving-step as an item, the proposed methods outperform the other methods including the CFAR and the TFF is a promising method (we will report the Bayesian Knowledge Tracing models in the next table, when using skill as an item).

Table 11.2 presents overall results on three data sets when using skill (or KC, called I_6 in Table 3.3). In this table, we compare the proposed methods with the state-of-the-art BKT models as well as the CFAR. The BKT-EM runs rather slow. Since it was intractable on Bridge, we have not collected its result on this data set.

Taking into account the student/task effects or considering multiple relationship between the students and the tasks on factorization models show promising results on these data. Moreover, in these results, when using skill (I_6) as an item, the tensor factorization or forecasting methods have less benefit than when using I_7 as an item. The reason for this could be because using the solving-step/problem (I_7) as an item may capture the *sequence of problem solving* better than using the skill as an item (each skill includes several solving-steps/problems, thus, causing ambiguity for the tensor/forecasting models). These results also validate what we have presented in Chapter 3, that, choosing the task as an item is also an important issue, and that different mappings may yield different results.

Overall, except the kNN methods on the ASSISTments data set, all the other proposed methods outperform the state-of-the-art BKT-BF and BKT-EM, as well as the CFAR.

Table 11.2: Overall comparison: Using skill (KC) as an item (the methods are ranked by using RMSE)

Algebra Data Set		Bridge Data Set		ASSISTments Data Set	
Method	RMSE	Method	RMSE	Method	RMSE
BMF	0.2996	BMF	0.2955	BPSEF	0.4326
WMRMF	0.3004	WMRMF	0.2963	WMRMF	0.4331
MRMF	0.3004	B-PDMF	0.2966	BPDEF	0.4334
TFF	0.3016	MRMF	0.2971	BPDMF	0.4338
TFMAF	0.3019	B-PDEF	0.2978	BMF	0.4346
TFW	0.3021	B-PSEF	0.2983	MRMF	0.4359
Task-kNN-Pearson	0.3022	TFF	0.2986	MF	0.4361
Task-kNN-Cosine	0.3022	TFMAF	0.2995	TFF	0.4374
B-PDEF	0.3027	Task-kNN-Pearson	0.2995	TFW	0.4380
B-PSEF	0.3031	MF	0.2996	TFMAF	0.4382
MF	0.3035	Task-kNN-Cosine	0.2998	TFA	0.4389
B-PDMF	0.3041	TFW	0.2999	Biased-Student-Task	0.4391
TFA	0.3050	TFA	0.3002	Logistic Regression	0.4397
Student-kNN-Cosine	0.3070	Logistic Regression	0.3003	Student Average	0.4448
CFAR	0.3070	Biased Student-Task	0.3006	BKT-EM	0.4497
Student-kNN-Pearson	0.3073	Student-kNN-Cosine	0.3017	Task kNN-Cosine	0.4530
BKT-EM	0.3111	CFAR	0.3053	Task kNN-Pearson	0.4563
Logistic Regression	0.3132	Student-kNN-Pearson	0.3072	Student kNN-Cosine	0.4583
BKT-BF	0.3176	BKT-BF	0.3087	CFAR	0.4677
Biased Student-Task	0.3197	Student Average	0.3114	BKT-BF	0.4692
Student Average	0.3287	Global Average	0.3186	Student kNN-Pearson	0.4705
Global Average	0.3328	BKT-EM	n/a	Global Average	0.4705

11.2 Discussion

So far, we have proposed several different methods for the problem of predicting student performance. The first obvious question one could raise is that *"Which methods should I use?"*

We have already seen the overall comparisons of all methods in the tables 11.1 and 11.2. However, we would like to highlight that we have developed different models for aiming at different usage purposes. So, the answers may be depending on the purposes as well as the data that one has at hand.

For example, if one does not have enough meta data (e.g., having student ID and task ID) and those available data follow a sequence[1], we propose either using the personalized forecasting methods, or using the tensor factorization to encode the relative

[1] e.g., sequence of problem solving. See details about sequential data in (Dietterich, 2002)

time (sequence), or incorporating the forecasting methods into the latent factor models (e.g., the proposed tensor factorization forecasting).

On the other hand, if one has more (meta) data, and those data have some relationships which involve them together, we propose using the multi-relational factorization approach since one may get benefit from exploiting such relational data. In addition, taking into account the student/task effects for the latent factor models (e.g., the BMF) one also get more improvements.

Furthermore, if one considers the student performance prediction as a binary classification problem, one should check whether the class imbalance appears in those data. If so, one would have benefit for prediction results by dealing with the class imbalance problem when building the models.

The second question we could discuss is that *"Have the proposed methods outperformed the state-of-the-arts by chance?"*

Before answering this question, we raise several reasons that we have not performed statistical significance tests for the experiments: *i)* as already mentioned, we would like to simulate the prediction results by using a real system from the KDD Cup 2010 website; *ii)* we also would like to see how far our approach can improve compared to the other methods on this system; *iii)* up until now, the true labels of the test sets have not been published so we have used the RMSE scores reported by this system; and finally *iv)* the most important reason is that the proposed tensor factorization and personalized forecasting approaches have been developed for learning on sequential data. This means that the order of the data sets must follow a sequence (e.g., training on the previous problems and predicting/forecasting on the last problems). Thus, directly applying cross-validation on these methods obviously does not work.

Moreover, as we have already seen in Tables 11.1 and 11.2, we have experimented using three large data sets on two different items (solving-step and skill) and found that the proposed methods outperform the state-of-the-art on all of these experiments (excepting the kNNs on ASSISTments data set). To validate again these results, we now report the RMSE on the validation sets for some typical methods in Tables 11.3 and 11.4. These finding results are also consistent with the results that we have presented before. We therefore conclude that the proposed approaches do not outperform the state-of-the-art by chance. Nevertheless, we will research how to split the sequential data for making significance tests in the future.

Since we have worked on the KDD Challenge 2010 data, another natural question would be that *"How good our approach is compared to the other teams on the same KDD Challenge 2010 data sets?"*

We summarize the results on Table 11.5, which presents the RMSE of the best student teams[1], on the KDD Challenge 2010 Leader-board.

The ranked first and second teams used feature engineering and ensembling techniques. For example, Yu *et al.* (2010) expanded sparse and dense features by various

[1]It would be paired to refer to the student teams since the author of this thesis, of course, is a student

Table 11.3: RMSE: Overall comparison on validation sets, using solving-step as an item

Algebra Data Set		Bridge Data Set		ASSISTments Data Set	
Method	RMSE	Method	RMSE	Method	RMSE
B-PDMF	0.3086	B-PDMF	0.2857	TFA	0.3818
TFMAF	0.3087	TFF	0.2897	BPDMF	0.3860
TFF	0.3091	TFA	0.2898	TFW	0.3905
TFA	0.3091	TFW	0.2901	TFMAF	0.3924
TFW	0.3092	TFMAF	0.2903	BPSEF	0.3954
B-PSEF	0.3104	BMF	0.2906	TFF	0.3974
WMRMF	0.3105	WMRMF	0.2909	WMRMF	0.3977
BMF	0.3118	Student-kNN-Cosine	0.2916	Student-kNN-Cosine	0.4131
Student-kNN-Cosine	0.3140	MF	0.2937	BMF	0.4134
MF	0.3147	B-PSEF	0.3023	MF	0.4294
CFAR	0.3225	CFAR	0.3071	CFAR	0.4566

Table 11.4: RMSE: Overall comparison on validation sets, using skill as an item

Algebra Data Set		Bridge Data Set		ASSISTments Data Set	
Method	RMSE	Method	RMSE	Method	RMSE
BMF	0.2995	BMF	0.2946	BPDMF	0.4061
MF	0.3005	MF	0.2951	BPSEF	0.4123
WMRMF	0.3016	WMRMF	0.2974	WMRMF	0.4170
TFW	0.3033	B-PDMF	0.2976	BMF	0.4197
TFMAF	0.3034	TFW	0.2992	TFF	0.4205
TFF	0.3034	B-PSEF	0.2994	MF	0.4207
TFA	0.3038	TFF	0.3002	TFMAF	0.4224
Task-kNN-Cosine	0.3041	TFA	0.3003	TFA	0.4237
B-PSEF	0.3050	Task-kNN-Cosine	0.3009	Task kNN-Cosine	0.4276
B-PDMF	0.3058	TFMAF	0.3021	TFW	0.4281
CFAR	0.3086	Student-kNN-Cosine	0.3027	Student kNN-Cosine	0.4480
Student-kNN-Cosine	0.3088	CFAR	0.3058	BKT-EM	0.4666
BKT-EM	0.3108	BKT-BF	0.3101	CFAR	0.4677
BKT-BF	0.3170	BKT-EM	n/a	BKT-BF	0.4879

binarization and discretization techniques, finally, they came up with millions of features and many models for making an ensemble; while Pardos & Heffernan (2010) used variants of the Bayesian Knowledge Tracing and feature engineering (called user features and skill features), finally, combining them using Random Forests ensemble.

As presented at the beginning, *the purpose of this thesis is not producing a system for the KDD Challenge* but on getting the public data sets from the real tutoring systems (Cognitive Tutor, Koedinger *et al.* (2010)), and proposing a new approach for

Table 11.5: RMSE on the leaderboard - KDD Challenge 2010 - Best student teams

Rank	Team Name	Algebra	Bridge	Average
1	National Taiwan University [Yu *et al.* (2010)]	0.2746	0.2713	0.2730
2	Zach A. Pardos [Pardos & Heffernan (2010)]	0.2778	0.2754	0.2766
-	**Our approach**	0.2803	0.2777	0.2790
3	SCUT Data Mining [Shen *et al.* (2010)]	0.2828	0.2781	0.2805
4	Y10 [Tabandeh & Sami (2010)]	0.3027	0.2933	0.2980

Student Modeling in an Intelligent Tutoring System, as well as bringing the state-of-the-art recommender system techniques in e-commerce domain to e-learning domain (specifically, to educational data mining and intelligent tutoring systems). Please also note that the result from our approach in Table 11.5 is just an off-line one and it was also combined from all the proposed methods[1]. It is well-known that an ensemble model works better than any single best model and we just would like to show that by making ensemble, the proposed approach can also be nearly approaching to the best teams, e.g. its average RMSE is just marginally higher than the first and the second team, for 0.006 and 0.0024, respectively.

Although the top team reached lower RMSE, these works require much human effort on data preprocessing as well as requiring intensive-memory of the computer. On the other hand, the proposed methods, e.g. personalized forecasting or factorization methods, are simple to implement and need not much human effort and computer memory to deal with large data sets (e.g., we just need two features for (biased) matrix factorization or personalized forecasting models, and three features for the tensor factorization models). However, more complex models using matrix factorization and ensembling them together could also produce better results, e.g., as shown in Töscher & Jahrer (2010), who won the third place overall (not a student team). This also means that factorization techniques are promising for the problem of predicting student performance.

11.3 Future Direction

As mentioned at the beginning, the use of recommender system techniques for student modeling is still a new topic. Although we have proposed several methods to approach this problem, several points are still open for future work, as outlined in the followings.

- **Model improvements**
  - Time-aware models for predicting student performance: From biology point of view, human biological clock (body-clock) may change over time for adapt-

[1]We have used the proposed methods to produce different results by changing different hyper parameters and by choosing different tasks as items, finally, we have combined them using a stacking ensemble. A total of 200 models were used.

ing with new time/environments. Consequently, the students would have higher probability to solve the problems correctly in the morning time (e.g. 8:30am - 12:30am) since they probably have good mood and feeling. Thus, one can change the error function by employing a weighting strategy, e.g., putting higher weight to the correct solutions at that time.

In contradictory, the students would not gain much knowledge at noon (e.g., 12:00pm - 2:00pm) or late afternoon (e.g. 17:30pm - 19:00pm) since they probably get tired after one day studying/working, and thus, they may have higher probability to make a mistake when solving the problem. Therefore, one can set a lower weight for correct solutions (or higher weight for the incorrect ones).

- Looking for a solution to model the slip and guess factors explicitly. For instance, as already discussed in Section 5.1, we have just expected that these factors are implicitly encoded in the latent models.
- Improving the loss function: Currently, we have used the squared loss function for all proposed methods, e.g., MF, TFF, WMRMF, etc. However, the data collected from the current tutoring systems are usually binary target variable (we also used binary relations in (W)MRMF), thus, adapting their loss functions (for binary target variable) may improve the prediction results.
- Finding other solutions to improve the WMRMF, e.g., weighting by the number of instances in each relation, or more sophisticated method as in Factorization Machines (Rendle, 2010), which implicitly learn to know which relations are important for predicting the target relation, instead of using a hyper parameter weight value.

  Furthermore, we also think about how to take into account both the multi-relational aspect and the sequential/temporal effect, e.g., by incorporating the forecasting techniques into the (W)MRMF models.
- Dealing with multiple skills. We have used only one skill (by simply considering a string of skills associated with the task as a single one), however, exploiting multiple skills may archive further improvements. For example, we can represent multiple skills in an item-skill matrix, then applying the the proposed (W)MRMF.

  For instance, a new ID is used for all empty skill values (≈20% empty values in the KDD Cup 2010 data), however, we can find a better way to deal with them, e.g., predicting these empty skills by using the similarities between the tasks. For example, if Task 1 requires Skill 1 and Skill 2; and Task 2 is similar to Task 1, so we can expect that Task 2 requires Skill 1 and Skill 2, too.
- Dealing with the cold-start problem: A challenging problem for using the collaborative filtering methods in educational environment is to deal with the new tasks or new students. We have simply used the global average

for alleviating this cold-start problem, however, more sophisticated methods may improve the prediction results.

- **Model validations**

  Validations on the data from the other fields should also be done to check whether the proposed methods can work on them. For instance, both the KDD Cup 2010 data and the ASSISTments Platform data involve math related problems. Also, as discussed in previous section, the significance tests should be applied for all proposed methods (except the methods in the class imbalance learning since we have already done that).

- **New model constructions**

  - Semi-supervised learning for predicting student performance: In many real-world applications, collecting the true labels is quite expensive while we can easily collect a plentiful amount of unlabeled data. For example, from the KDD Cup 2010 data, we are provided a large amount on the test set but without the labels. Using semi-supervised learning to utilize these unlabeled data would have benefit in prediction results.
  - Utilizing both the Hidden Markov Models (HMM) and Latent Factor Models for predicting student performance (or generally, for modeling the sequential data) could be an approach for future study. A similarly recent work for learning HMM using non-negative matrix factorization can be found in Cybenko & Crespi (2011).
  - Since there is a sequence in student problem solving/learning, finding a way for modeling sequential data, e.g. in Bengio (1999), would also be a choice.

- **Applications**

  So far, many methods have been proposed, however, we still lack of the real applications to employ these methods into real environments. Some of the applications could be:

  - Building recommender systems for student's task recommendation, as already discussed in Chapter 9. In the same way, we could also build recommendation systems to support the instructors (for the Tutoring Model in an ITS, as in Figure 1.2) to recommend the instructors which tasks should be constructed to teach the students.
  - Building recommender systems using state-of-the-art methods to recommend learning resources to the students. Moreover, since mobile devices (smart phones, tablet PC, etc) are becoming popular, utilizing them for e-learning can provide benefit. For example, one application could be "mobile context-aware system for student resource recommendation". With this, students could navigate the learning resources (books, seminars, group discussions, etc) anywhere at anytime.

- Integrating recommender systems into some existing e-learning systems, e.g. the Moodle, to help the students better finding resources.
- Integrating a (local) social network system into an ITS. By doing so, we may also understand better about how the students share their knowledge. For example, if A is a clever student, he is also interested in studying; and if B and C are A's friends, we would expect that B and C can somehow learn from A's knowledge. Moreover, the data getting from such systems could be utilized by social recommender system techniques, e.g., social matrix factorization (Jamali & Ester, 2010), thus, we may get better results in predicting student performance.

We hope that the contributions and the open issues in this work would inspire the researchers in these domains for further improvements.

Index

Algebra, 19
ASSISTments, 19, 20

B-PDEF, 74
B-PDMF, 76
B-PSEF, 74
B-PSEF-SGD, 75
B42, 125
BAN, 118
BANOpt, 120
Baseline predictor, 49
Baselines, 50
Bayes risk, 107
Bayesian Knowledge Tracing, 30
Bayesian Network Augmented Naive Bayes, 118
Bayesian Networks, 118
Beta function, 124
Biased Matrix Factorization, 45
Biased Personalized Discounted-Mean Forecaster, 76
Biased Personalized Double Exp. smoothing Forecaster, 74
Biased Personalized Single Exp. smoothing Forecaster, 74
Biased student-task, 49–51, 78
Binary-class classification, 15
BKT, 30
BKT-BF, 32, 51
BKT-CGS, 35
BKT-EM, 31, 51
BKT-PPS, 34
BMF, 45
BNCost, 121
BNOpt, 120
Bootstrapping, 77
Borderline examples, 104
Bridge, 19

CANDECOM-PARAFAC, 85
CFA, 20
CFAR, 29, 49
Class imbalance problem, 15, 102, 103
Classification problem, 15
Classifier-Level Approach, 106
Combination Approach, 107
Concrete time, 84, 96
Conditional probability tables, 119
Conditional risk, 107
Context-aware factorization models, 97
Contextual modeling, 98
Convergence, 42
Correct first attempt, 20–24, 91
Cosine similarity, 48
Cost ratio, 107
Cost-sensitive learning, 106
CPT, 119
CSL, 106, 128
CSW, 128

Data encoding, 25
Data sequence, 84
Data splitting, 26
Data splitting with temporal constraint, 26
Data-Level Approach, 104
DEF, 69
Domain Knowledge, 2
Domain Model, 2
Double Exp. Smoothing Forecaster, 69

Double exponential smoothing, 69

ERD, 63

F-Measure, 109
False Positive Rate, 109
Forecasting problem, 15
Forecasting using history length, 70
Frobenius norm, 43

Global average, 46, 50
GMean, 109, 116
Gradient, 42
Gradient descent, 42, 43, 75, 78
Guess, 5, 31, 83

histLength, 70, 79
History length, 70–73, 75, 79, 86, 88, 89, 93

Imbalanced data, 103
Imbalanced ratio, 103
Implementations, 49
Intelligent Tutoring Systems, 1
item average, 24
Item bias, 45
Item effect, 45
Item recommendation, 4
Item-based Collaborative Filtering, 48
Item-kNN, 48
ITS, 1

k-step-ahead forecasting, 77
KC, 19
Kernel functions, 113
kNN, 47
Knowledge component, 13, 19

Latent factor models, 40
Latent factors, 41
Leader-board, 152
Learn-BNOpt, 120
Learn-EnsBNOpt, 120
Learn-S-BNOpt, 123
Learning Environment, 3
Learning Factor Analysis, 35
Learning rate, 42
LFA, 35

MAE, 14
Majority class, 25, 102, 103
Markov Blanket Bayesian Classifier, 118
Matrix factorization, 41
MBOpt, 120
MetaCost, 107, 128
MF, 41
Minority class, 25, 102, 103
Moving average, 68
MRMF, 57, 60
Multi-class classification, 15
Multi-Relational Matrix Factorization, 60
Multiple interactions, 95
Multiple skills, 25, 155

N-PDEF, 74
N-PSEF, 72
N-PSEF-SGD, 72
Naive Bayes, 118
Negative class, 103
new-item problem, 22
Non-biased Personalized Double Exp. smoothing Forecaster, 74
Non-biased Personalized Single Exp. smoothing Forecaster, 72
None-Context approach, 97
Normal distribution, 42
Number of latent factors, 41, 62, 86

O-CSL, 112, 115
Opportunity count, 19
Optimized cost ratio, 115
Over-fitting, 43

Pearson similarity, 48
Performance Factors Analysis (PFA), 35
Personalized forecasting, 68, 72
PFA, 35
Polynomial kernel, 113

Positive class, 103
Pre-Filtering, 98
Precision, 109
Predicting student performance, 5, 13, 85
Problem hierarchy, 19
Problem name, 19
Problem view, 19, 96
Problem-step, 12, 53
PSP, 5

Radial basis function, 113
Random oversampling, 104
Random undersampling, 104
Rating prediction, 4, 14
Recall, 109
Recommender Systems, 4
Regression problem, 14
Regularization, 43
Regularization term, 43
regularized logistic regression, 51
Relation weight, 61
Relative time, 84, 98
RMSE, 14, 41
Root Mean Squared Error, 14
ROS, 104
RS, 4
RUS, 104

S-CSL, 112
secLength, 70, 79
Section, 19
Section length, 70
SEF, 69
Sequence, 85
Sequential data, 84
Sequential effect, 84
Similarity, 48
Single Exp. Smoothing Forecaster, 69
Single exponential smoothing, 68
Skill, 13, 19
Slip, 5, 31, 84
SMOTE, 104
Softwares, 49
solving-step, 52
sparsity, 22, 23
Step, 19
Stochastic gradient descent, 43, 44, 47, 60, 72, 73, 75, 88, 89, 97, 99
Student average, 50
Student bias, 45, 74
Student correlation, 47
Student effect, 7, 45, 147
Student latent factor, 86
Student Model, 2
Student performance, 13
Student-kNN, 48
SVM, 112

TAN, 118
TANOpt, 120
Task, 12
Task bias, 45, 46, 50, 74, 87, 90
Task correlation, 47
Task effect, 7, 45, 147
Task latent factor, 86
Task recommendation, 95
Task-kNN, 49
Temporal effect, 84
Temporal effects, 95
Tensor, 84
Tensor decomposition, 85
Tensor Factorization - Averaging, 87
Tensor Factorization - Moving Average Forecasting, 88
Tensor Factorization - Weighting, 87
Tensor Factorization Forecasting, 88
Tensor mode, 84
TFA, 87
TFF, 88
TFMAF, 88
TFW, 87
Three-mode tensor, 85
TLINK, 104
TNR, 109
Tomek's Link, 104
Total costs, 107, 115

TPR, 109
Tree Augmented Naive Bayes, 118
True Negative Rate, 109
True Positive Rate, 109
Tutoring Model, 3
Two-mode tensor, 84

Unit, 19
User bias, 45
User effect, 45
User Interface, 3
User-based collaborative filtering, 48
User-item baseline, 49, 50
User-kNN, 48

Weighted Multi-Relational Matrix Factorization, 61
WMRMF, 58, 61, 149, 150

References

Adomavicius, G. & Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, **17**, 734–749. 49

Adomavicius, G. & Tuzhilin, A. (2011). Context-aware recommender systems. In F. Ricci, L. Rokach, B. Shapira & P.B. Kantor, eds., *Recommender Systems Handbook*, 217–253, Springer. 84, 96, 97, 98

Akbani, R., Kwek, S. & Japkowicz, N. (2004). Applying support vector machines to imbalanced datasets. In *Proceedings of the 15th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*, 39–50. 107

Baker, R., Pardos, Z., Gowda, S., Nooraei, B. & Heffernan, N. (2011). Ensembling predictions of student knowledge within intelligent tutoring systems. In J. Konstan, R. Conejo, J. Marzo & N. Oliver, eds., *User Modeling, Adaption and Personalization*, vol. 6787 of *Lecture Notes in Computer Science*, 13–24, Springer Berlin / Heidelberg. 32, 36, 37, 52

Baker, R.S. & Yacef, K. (2009). The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining (JEDM)*, **1**, 3–17. 27

Baker, R.S., Corbett, A.T. & Aleven, V. (2008a). Improving contextual models of guessing and slipping with a truncated training set. In *The first International Conference on Educational Data Mining (EDM 2008)*, 67–76. 36

Baker, R.S., Corbett, A.T. & Aleven, V. (2008b). More accurate student modeling through contextual estimation of slip and guess probabilities in bayesian knowledge tracing. In *Proceedings of the 9th International Conference on Intelligent Tutoring Systems*, ITS '08, 406–415, Springer-Verlag, Berlin, Heidelberg. 32, 33, 36, 37, 38, 40, 50, 51, 54, 65, 78, 81, 92

Baker, R.S.J., Corbett, A.T., Gowda, S.M., Wagner, A.Z., MacLaren, B.A., Kauffman, L.R., Mitchell, A.P. & Giguere, S. (2010). Contextual slip and prediction of student performance after use of an intelligent tutor. In *the 18th*

International Conference on User Modeling, Adaption and Personalization (UMAP 2010), Lecture Notes in Computer Science, 52–63, Springer. 35, 36

Beck, J., Stern, M. & Haugsjaa, E. (1996). Applications of ai in education. *Crossroads*, **3**, 11–15. 1, 2, 3, 13

Beck, J.E. & Chang, K.M. (2007). Identifiability: A fundamental problem of student modeling. In *Proceedings of the 11th international conference on User Modeling*, UM '07, 137–146, Springer-Verlag, Berlin, Heidelberg. 32, 36

Bekele, R. & Menzel, W. (2005). A bayesian approach to predict performance of a student (bapps): A case with ethiopian students. In *Proceedings of the International Conference on Artificial Intelligence and Applications*, 189–194, Vienna, Austria. 27

Bell, R.M. & Koren, Y. (2007). Scalable collaborative filtering with jointly derived neighborhood interpolation weights. In *Proceedings of the 7th IEEE International Conference on Data Mining (ICDM 2007)*, 43–52, IEEE Computer Society, Washington, USA. 38, 40, 42

Bengio, Y. (1999). Markovian models for sequential data. *Neural Computing Surveys*, **2**, 129–162. 84, 156

Bobadilla, J., Serradilla, F. & Hernando, A. (2009). Collaborative filtering adapted to recommender systems of e-learning. *Knowledge-Based Systems*, **22**, 261–265. 4, 96

Bottou, L. (2004). Stochastic learning. In O. Bousquet & U. von Luxburg, eds., *Advanced Lectures on Machine Learning*, Lecture Notes in Artificial Intelligence, 146–168, Springer Verlag, Berlin. 43, 97

Box, G.E.P. & Jenkins, G. (1990). *Time Series Analysis, Forecasting and Control*. Holden-Day, Incorporated. 68, 69

Bradley, A.P. (1997). The use of the area under the ROC curve in the evaluation of machine learning algorithms. *Pattern Recognition*, **7**, 1145–1159. 102, 109

Breiman, L. (2001). Random forests. *Mach. Learn.*, **45**, 5–32. 29

Brockwell, P.J. & Davis, R.A. (2002). *Introduction to Time Series and Forecasting*. Springer. 68, 69, 77, 78

Carroll, J. & Chang, J.J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of eckart-young decomposition. *Psychometrika*, **35**, 283–319. 85

Cen, H. (2009). *Generalized Learning Factors Analysis: Improving cognitive Models with Machine Learning*. Ph.D. thesis, Machine Learning Department, School of Computer Science, Carnegie Mellon University, USA. 5

Cen, H., Koedinger, K. & Junker, B. (2006). Learning factors analysis a general method for cognitive model evaluation and improvement. In *Proceeding of the International Conference on Intelligent Tutoring Systems*, vol. 4053, 164–175, Springer Berlin Heidelberg. 35

Cen, H., Koedinger, K.R. & Junker, B. (2007). Is over practice necessary? – improving learning efficiency with the cognitive tutor through educational data mining. In *Proceeding of the 2007 Conference on Artificial Intelligence in Education: Building Technology Rich Learning Contexts That Work (AIED 2007)*, 511–518, IOS Press, Amsterdam, The Netherlands, The Netherlands. 147

Cetintas, S., Si, L., Xin, Y. & Hord, C. (2010). Predicting correctness of problem solving in intelligent tutoring systems with a temporal collaborative filtering approach. In *Proceedings of the International Conference on Intelligent Tutoring Systems (ITS 2010)*, 15–24. 96

Chai, X., Deng, L., Yang, Q. & Ling, C.X. (2004). Test-cost sensitive naive bayes classification. In *Proceedings of the IEEE International Conference on Data Mining (ICDM 2004)*, 51–58, IEEE Computer Society, Los Alamitos, CA, USA. 107

Chang, K., Beck, J., Mostow, J. & Corbett, A. (2006). A bayes net toolkit for student modeling in intelligent tutoring systems. In *Proceedings of International Conference on Intelligent Tutoring Systems (ITS 2006)*, 104–113, Springer. 31, 32, 33, 36, 37, 38, 40, 50, 51, 54, 65, 78, 81, 92

Chatfield, C. & Yar, M. (1988). Holt-winters forecasting: Some practical issues. *Special Issue: Statistical Forecasting and Decision-Making. Journal of the Royal Statistical Society. Series D (The Statistician)*, **37**, 129–140. 69, 93

Chawla, N.V., Bowyer, K., Hall, L. & Kegelmeyer, W.P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligent Research*, **16**, 321–357. 104, 105, 107, 108, 128

Chawla, N.V., Japkowicz, N. & Kotcz, A. (2004). Editorial: special issue on learning from imbalanced data sets. *SIGKDD Explorations*, **6**, 1–6. 25, 102, 104, 117

Chen, X.W., Gerlach, B. & Casasent, D. (2005). Pruning support vectors for imbalanced data classification. In *Proceeding of the IEEE International Joint Conference on Neural Networks (IJCNN 2005)*, 1883–1888, IEEE Xplore. 108

Cheng, J. & Greiner, R. (1999). Comparing bayesian network classifiers. In *Proceedings of Conference on Uncertainty in Artificial Intelligence (UAI'99)*. 117, 118

Chi, M., Koedinger, K., Gordon, G., Jordan, P. & Vanlehn, K. (2011). Instructional factors analysis: A cognitive model for multiple instructional interventions. In *Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and*

Stamper, J. (Eds.). Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011), 61–70. 36

Cichocki, A., Zdunek, R., Phan, A.H. & ichi Amari, S. (2009). *Nonnegative Matrix and Tensor Factorizations - Applications to Exploratory Multi-way Data Analysis and Blind Source Separation*. Wiley. 86

Cieslak, D.A. & Chawla, N.V. (2008). Learning decision trees for unbalanced data. In *ECML PKDD '08: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases*, 241–256. 108

Conati, C. (2010). Bayesian student modeling. In R. Nkambou, J. Bourdeau & R. Mizoguchi, eds., *Advances in Intelligent Tutoring Systems*, vol. 308 of *Studies in Computational Intelligence*, 281–299, Springer Berlin / Heidelberg. 36

Cooper, G.F. & Herskovits, E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, **9**, 309–347. 119

Corbett, A.T. & Anderson, J.R. (1995). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, **4**, 253–278. 3, 5, 30, 33, 38, 40, 50, 51, 65, 78, 81, 84, 129

Corbett, A.T., Koedinger, K.R. & Anderson, J.R. (1997). Intelligent tutoring systems. In M.G. Helander, T.K. Landauer & P.V. Prabhu, eds., *Handbook of human-computer interaction*, 849–874, Elsevier Science B. V., Amsterdam. 1, 2

Cortes, C. & Vapnik, V. (1995). Support-vector networks. *Machine Learning*, **20**, 273–297. 112

Cybenko, G. & Crespi, V. (2011). Learning hidden markov models using nonnegative matrix factorization. *IEEE Transactions on Information Theory*, **57**, 3963–3970. 156

Degroot, M.H. & Schervish, M.J. (2002). *Probability and Statistics, 3rd Edition*. Addison-Wesley. 124

Delavari, N., Beikzadeh, M.R. & Shirazi, M.R.A. (2004). A new model for using data mining in higher educational system. *5th International Conference on Information Technology Based Higher Education and Training*. 27

Desmarais, M. & Baker, R. (2011). A review of recent advances in learner and skill modeling in intelligent learning environments. *User Modeling and User-Adapted Interaction*, 1–30. 5, 27, 36

Dietterich, T.G. (2002). Machine learning for sequential data: A review. In *Proceedings of the Joint IAPR International Workshop on Structural, Syntactic, and Statistical Pattern Recognition*, 15–30, Springer-Verlag, London, UK. 84, 151

Domingos, P. (1999). Metacost: A general method for making classifiers cost-sensitive. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data mining (KDD 1999)*, 155–164, San Diego CA USA. 106, 107, 108, 112, 115, 117, 128

Drachsler, H., Hummel, H.G.K. & Koper, R. (2009). Identifying the goal, user model and conditions of recommender systems for formal and informal learning. *Journal of Digital Information*, **10**, 4–24. 95, 96

Duda, R., Hart, P. & Stork, D. (2000). *Pattern Classification*. Wiley-Interscience Publication. 2

Dunlavy, D.M., Kolda, T.G. & Acar, E. (2011). Temporal link prediction using matrix and tensor factorizations. *ACM Trans. Knowl. Discov. Data*, **5**, 10:1–10:27. 86, 93

Elkan, C. (2001). The foundations of cost-senstive learning. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI'01)*, 973–978, Seattle - WA. 106, 107, 108, 117, 121, 128

Fan, R.E., Chang, K.W., Hsieh, C.J., Wang, X.R. & Lin, C.J. (2008). Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, **9**, 1871–1874. 29, 128

Fan, W., Greengrass, E., McCloskey, J., Yu, P.S. & Drummey, K. (2005). Effective estimation of posterior probabilities: Explaining the accuracy of randomized decision tree approaches. *IEEE International Conference on Data Mining (ICDM 2005)*, 154–161. 117, 121

Feng, M. (2009). *Towards Assessing Students' Fine Grained Knowledge: Using an Intelligent Tutor for Assessment*. Ph.D. thesis, Computer Science Department, Worcester Polytechnic Institute, USA. 5

Feng, M., Heffernan, N. & Koedinger, K. (2009). Addressing the assessment challenge with an online system that tutors as it assesses. *User Modeling and User-Adapted Interaction*, **19**, 243–266. 5, 18, 20, 21, 147

Freedman, R., Ali, S.S. & McRoy, S. (2000). Links: what is an intelligent tutoring system? *Intelligence*, **11**, 15–16. 2

Friedman, N., Geiger, D. & Goldszmidt, M. (1997). Bayesian network classifiers. *Machine Learning*, **29**, 131–163. 117, 118, 134

Gantner, Z., Drumond, L., Freudenthaler, C., Rendle, S. & Schmidt-Thieme, L. (2010a). Learning attribute-to-feature mappings for cold-start recommendations. In *Proceedings of the 10th IEEE International Conference on Data Mining (ICDM 2010)*, IEEE Computer Society. 52

Gantner, Z., Rendle, S. & Schmidt-Thieme, L. (2010b). Factorization models for context-/time-aware movie recommendations. In *Proceedings of the Workshop on Context-Aware Movie Recommendation*, CAMRa '10, 14–19, ACM, New York, NY, USA. 84

Gantner, Z., Rendle, S., Freudenthaler, C. & Schmidt-Thieme, L. (2011). Mymedialite: A free recommender system library. In *Proceedings of the 5th International Conference on Recommender Systems (RecSys 2011)*, ACM, Chicago, USA. 50

Garcia, E., Romero, C., Ventura, S. & Castro, C.D. (2009). An architecture for making recommendations to courseware authors using association rule mining and collaborative filtering. *User Modeling and User-Adapted Interaction*, **19**. 96

Ge, L., Kong, W. & Luo, J. (2006). Courseware recommendation in e-learning system. In *International Conference on Web-based Learning (ICWL'06)*, 10–24. 96

Ghauth, K. & Abdullah, N. (2010). Learning materials recommendation using good learners' ratings and content-based filtering. *Educational Technology Research and Development, Springer-Boston*, 1042–1629. 4, 96

Gong, Y. & Beck, J. (2011). Looking beyond transfer models: Finding other sources of power for student models. In J. Konstan, R. Conejo, J. Marzo & N. Oliver, eds., *User Modeling, Adaption and Personalization*, vol. 6787 of *Lecture Notes in Computer Science*, 135–146, Springer Berlin / Heidelberg. 36

Gong, Y., Beck, J. & Heffernan, N. (2010a). Comparing knowledge tracing and performance factor analysis by using multiple model fitting procedures. In V. Aleven, J. Kay & J. Mostow, eds., *Intelligent Tutoring Systems*, vol. 6094 of *Lecture Notes in Computer Science*, 35–44, Springer Berlin / Heidelberg. 30, 32, 36, 37, 52

Gong, Y., Beck, J. & Heffernan, N. (2010b). How to construct more accurate student models: Comparing and optimizing knowledge tracing and performance factor analysis. *International Journal of Artificial Intelligence in Education*. 37

Gowda, S., Baker, R.S., Pardos, Z. & Heffernan, N. (2011a). The sum is greater than the parts: Ensembling student knowledge models in assistments. In *Proceedings of the KDD 2011 Workshop on Knowledge Discovery in Educational Data (KDDinED 2011). Held as part of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011)*, San Diego, CA, USA. 37

Gowda, S.M., Rowe, J.P., Baker, R.S., Chi, M. & Koedinger, K.R. (2011b). Improving models of slipping, guessing, and moment-by-moment learning with estimates of skill difficulty. In *Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and Stamper, J. (Eds.). Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011)*, 199–208, Eindhoven, the Netherlands. 36

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P. & Witten, I.H. (2009). The weka data mining software: An update. *SIGKDD Exploration*, **11**. 118, 119, 120

Hand, D.J. (2006). Classifier technology and the illusion of progress. *Statistical Science*, **21**, 1–14. 110, 116

Hand, D.J. (2009). Measuring classifier performance: A coherent alternative to the area under the ROC curve. *Machine Learning*, **77**, 103–123. 102, 110, 111, 116, 125, 143

Hanley, J. & Mcneil, B. (1982). The meaning and use of the area under receiver operating characteristics (ROC) curve. *Radiology*, **143**, 29–36. 102

Harshman, R.A. (1970). Foundations of the PARAFAC procedure: Models and conditions for an explanatory multi-modal factor analysis. *UCLA Working Papers in Phonetics*, **16**, 84. 85

Hart, P.E. (1968). The condensed nearest neighbor rule. *IEEE Trans. Inf. Theory*, 515–516. 104

He, H. & Garcia, E.A. (2009). Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, **21**, 1263–1284. 25, 102, 104, 108, 117, 124

Herlocker, J.L., Konstan, J.A., Borchers, A. & Riedl, J. (1999). An algorithmic framework for performing collaborative filtering. In *Proceedings of the 22nd ACM International Conference on Research and Development in Information Retrieval (SIGIR 1999)*, 230–237, ACM, New York, NY, USA. 49

Hido, S. & Kashima, H. (2008). Roughly balanced bagging for imbalanced data. In *Proceedings of the SIAM International Conference on Data Mining (SDM'08)*, 143–152. 108, 109, 116, 117, 121

Hsu, C.W., Chang, C.C. & Lin, C.J. (2003). *A practical guide to support vector classification*. Department of Computer Science and Information Engineering, National Taiwan University. 113

Jamali, M. & Ester, M. (2010). A matrix factorization technique with trust propagation for recommendation in social networks. In *Proceedings of the fourth ACM Conference on Recommender Systems*, RecSys '10, 135–142, ACM, New York, NY, USA. 157

Jannach, D., Zanker, M., Felfernig, A. & Friedrich, G. (2011). *Recommender Systems An Introduction*. Cambridge University Press. 4

Karatzoglou, A., Amatriain, X., Baltrunas, L. & Oliver, N. (2010). Multiverse recommendation: n-dimensional tensor factorization for context-aware collaborative filtering. In *Proceedings of the fourth ACM conference on Recommender systems*, RecSys '10, 79–86, ACM, New York, NY, USA. 86

Khribi, M.K., Jemni, M. & Nasraoui, O. (2008). Automatic recommendations for e-learning personalization based on web usage mining techniques and information retrieval. In *Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies*, 241–245, IEEE Computer Society. 96

Kittler, J., Hatef, M., Duin, R.P. & Matas, J. (1998). On combining classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **20**, 226–239. 120

Klement, W., Wilk, S., Michaowski, W. & Matwin, S. (2009). Dealing with severely imbalanced data. *Workshop on Data Mining When Classes are Imbalanced and Errors Have Costs in Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'09)*. 108, 117

Koedinger, K., Baker, R., Cunningham, K., Skogsholm, A., Leber, B. & Stamper, J. (2010). A data repository for the edm community: The pslc datashop. In C. Romero, S. Ventura, M. Pechenizkiy & R. Baker, eds., *Handbook of Educational Data Mining*, Lecture Notes in Computer Science, CRC Press. 5, 13, 16, 18, 20, 26, 54, 153

Kolda, T.G. & Bader, B.W. (2009). Tensor decompositions and applications. *SIAM Review*, **51**, 455–500. 84, 86, 97, 99

Komarek, P. & Moore, A.W. (2005). Making logistic regression a core data mining tool with tr-irls. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM 2005)*, ICDM '05, 685–688, IEEE Computer Society, Washington, DC, USA. 38, 50, 51

Koren, Y. (2010). Factor in the neighbors: Scalable and accurate collaborative filtering. *ACM Transactions on Knowledge Discovery from Data*, **4**, 1–24. 38, 60, 78

Koren, Y. & Bell, R.M. (2011). Advances in collaborative filtering. In F. Ricci, L. Rokach, B. Shapira & P.B. Kantor, eds., *Recommender Systems Handbook*, 145–186, Springer. 49, 50

Koren, Y., Bell, R. & Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *IEEE Computer Society Press*, **42**, 30–37. 38, 40, 41, 45

Kotsiantis, S.B. & Pintelas, P.E. (2005). Predicting students marks in hellenic open university. In *IEEE International Conference on Advanced Learning Technologies (ICALT 2005)*, 664–668. 27

Kubat, M. & Matwin, S. (1997). Addressing the curse of imbalanced training sets: One-sided selection. In *Proceedings of the 14th International Conference on Machine Learning (ICML 1997)*, 179–186, Morgan Kaufmann. 106, 109, 116, 132

Kuncheva, L.I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience. 120

Lawrence, N.D. & Urtasun, R. (2009). Non-linear matrix factorization with gaussian processes. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, 601–608, ACM, New York, NY, USA. 41

Lessmann, S. (2004). Solving imbalanced classification problems with support vector machines. In *International Conference on Artificial Intelligence*, 214–220. 108

Li, P., Qiao, P.L. & Liu, Y.C. (2008a). A hybrid re-sampling method for SVM learning from imbalanced data sets. In *Proceedings of the 2008 Fifth IC on Fuzzy Systems and Knowledge Discovery*, 65–69, IEEE Computer Society, Washington, DC, USA. 104

Li, X., Wang, L. & Sung, E. (2008b). AdaBoost with SVM-based component classifiers. *Engineering Applications of Artificial Intelligence*, **21**, 785–795. 108

Li, Y. & Zhang, X. (2011). Improving k nearest neighbor with exemplar generalization for imbalanced classification. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2011)*, 321–332, Springer. 108

Linden, G., Smith, B. & York, J. (2003). Amazon.com recommendations: Item-to-item collaborative filtering. *IEEE Internet Computing*, **7**, 76–80. 4

Lippert, C., Weber, S.H., Huang, Y., Tresp, V., Schubert, M. & Kriegel, H.P. (2008). Relation-prediction in multi-relational domains using matrix-factorization. In *NIPS 2008 Workshop: Structured Input - Structured Output*. 56, 57, 60, 61, 64

Liu, W., Chawla, S., Cieslak, D.A. & Chawla, N.V. (2010). A robust decision tree algorithm for imbalanced data sets. In *SIAM International Conference on Data Mining (SDM'10)*, 766–777. 108, 117

Liu, X.Y. & Zhou, Z.H. (2006). The influence of class imbalance on cost-sensitive learning: An empirical study. In *Proceedings of the Sixth IEEE International Conference on Data Mining (ICDM 2006)*, 970–974, IEEE Computer Society, Washington, DC, USA. 112, 115

Liu, X.Y., Wu, J. & Zhou, Z.H. (2009). Exploratory undersampling for class-imbalance learning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, **39**, 539–550. 107, 109, 116, 117

Luo, J., Dong, F., Cao, J. & Song, A. (2010). A context-aware personalized resource recommendation for pervasive learning. *Cluster Computing, Springer-Netherlands*, **13**, 213–239. 96

Madden, M.G. (2002). A new bayesian network structure for classification tasks. In *Proceedings of the 13th Irish International Conference on Artificial Intelligence and Cognitive Science*, 203–208, Springer-Verlag, London-UK. 118, 119, 134

Maloof, M.A. (2003). Learning when data sets are imbalanced and when costs are unequeal and unknown. *Workshop on Learning from Imbalanced Data Sets II in International Conference on Machine Learning (ICML'03)*. 117

Manouselis, N., Drachsler, H., Vuorikari, R., Hummel, H. & Koper, R. (2010). Recommender systems in technology enhanced learning. In *Kantor, P.B., Ricci, F., Rokach, L., Shapira, B. (eds.) 1st Recommender Systems Handbook*, 1–29, Springer-Berlin. 4, 96

Margineantu, D.D. (2000). When does imbalanced data require more than cost-sensitive learning? In *Workshop on Learning from Imbalanced Data, Artificial Intelligence (AAAI-2000)*. 115

Massey, L., Psotka, J. & Mutter, S.A. (1988). *Intelligent tutoring systems : lessons learned / edited by Joseph Psotka, L. Dan Massey, Sharon A. Mutter, advisory editor, John Seely Brown*. L. Erlbaum Associates, Hillsdale, N.J. 2, 12

Minaei-Bidgoli, B., Kashy, D.A., Kortemeyer, G. & Punch, W.F. (2003). Predicting student performance: an application of data mining methods with an educational web-based system. In *The 33rd IEEE Conference on Frontiers in Education (FIE 2003)*, 13–18. 27

Murphy, K.P. (2001). The bayes net toolbox for matlab. *Computing Science and Statistics*, **33**, 2001. 31

Nickerson, A., Japkowicz, N. & Millos, E. (2001). Using unsupervised learning to guide resampling in imbalanced data sets. In *Proceedings of the Eighth International Workshop on AI and Statitsics*, 261–265. 104

NIST (2010). *NIST/SEMATECH e-Handbook of Statistical Methods. http://www.itl.nist.gov/div898/handbook*. NIST. 68, 77

Nooraei, B., Pardos, Z., Heffernan, N. & Baker, R. (2011). Less is more: Improving the speed and prediction power of knowledge tracing by using less data. In *Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and Stamper, J. (Eds.). Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011)*, 101–110, Eindhoven, the Netherlands. 36

OMahony, M.P. & Smyth, B. (2007). A recommender system for on-line course enrolment: An initial study. In *ACM Conference on Recommender System (RecSys 2007)*, 973–978, Minnesota USA. 96

Pardos, Z. & Heffernan, N. (2011). Kt-idem: Introducing item difficulty to the knowledge tracing model. In J. Konstan, R. Conejo, J. Marzo & N. Oliver, eds., *User Modeling, Adaption and Personalization*, vol. 6787 of *Lecture Notes in Computer Science*, 243–254, Springer Berlin / Heidelberg. 36

Pardos, Z., Gowda, S., Baker, R.S. & Heffernan, N. (2011). Ensembling predictions of student post-test scores for an intelligent tutoring system. In *Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and Stamper, J. (Eds.). Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011)*, 189–198. 35, 37

Pardos, Z.A. & Heffernan, N.T. (2010). Using hmms and bagged decision trees to leverage rich features of user and skill from an intelligent tutoring system dataset. In *Proceedings of the KDD Cup 2010 Workshop on Improving Cognitive Models with Educational Data Mining*, Washington, DC, USA. ix, 25, 34, 36, 37, 40, 153, 154

Paterek, A. (2007). Improving regularized singular value decomposition for collaborative filtering. In *Proceedings of KDD Cup Workshop at the 13th ACM International Conference on Knowledge Discovery and Data Mining*, 39–42. 38

Pavlik, P., Yudelson, M. & Koedinger, K. (2011). Using contextual factors analysis to explain transfer of least common multiple skills. In G. Biswas, S. Bull, J. Kay & A. Mitrovic, eds., *Artificial Intelligence in Education*, vol. 6738 of *Lecture Notes in Computer Science*, 256–263, Springer Berlin / Heidelberg. 36

Pavlik, P.I., Cen, H. & Koedinger, K.R. (2009). Performance factors analysis –a new alternative to knowledge tracing. In *Proceeding of the 2009 conference on Artificial Intelligence in Education: Building Learning Systems that Care: From Knowledge Representation to Affective Modelling*, 531–538, IOS Press, Amsterdam, The Netherlands, The Netherlands. 35, 36

Pilászy, I. & Tikk, D. (2009). Recommending new movies: even a few ratings are more valuable than metadata. In *Proceedings of the third ACM Conference on Recommender Systems*, RecSys '09, 93–100, ACM, New York, USA. 40, 41

Platt, J.C. (1999a). *Fast training of support vector machines using sequential minimal optimization*, 185–208. MIT Press, Cambridge, MA, USA. 128

Platt, J.C. (1999b). Probabilistic outputs for SVM and comparisons to regularized likelihood methods. In *Advances in Large Margin Classifiers*, 61–74, MIT Press. 115

Prati, R.C., Batista, G.E.A.P.A. & Monard, M.C. (2004). Class imbalances versus class overlapping: An analysis of a learning system behavior. In *Advances in Artificial Intelligence*, 312–321. 132

Preisach, C., Marinho, L.B. & Schmidt-Thieme, L. (2010). Semi-supervised tag recommendation - using untagged resources to mitigate cold-start problems. In *14th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2010)*, vol. 6118 of *Lecture Notes in Computer Science*, 348–357, Springer. 52

Rabiner, L. & Juang, B. (1986). An introduction to hidden markov models. *IEEE Acoutics, Speech and Signal Processing Magazine*, **3**, 4–16. 31

Rabiner, L.R. (1990). A tutorial on hidden markov models and selected applications in speech recognition. In A. Waibel & K.F. Lee, eds., *Readings in speech recognition*, 267–296, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. 31

Rai, D., Gong, Y. & Beck, J.E. (2009). Using dirichlet priors to improve model parameter plausibility. In *2nd International Conference On Educational Data Mining*, 141–150. 36

Raskutti, B. & Kowalczyk, A. (2004). Extreme re-balancing for SVMs: a case study. *SIGKDD Explorations*, **6**, 60–69. 104, 115

Rendle, S. (2010). Factorization machines. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM 2010)*, 995–1000, IEEE Computer Society, Washington, DC, USA. 155

Rendle, S. & Schmidt-Thieme, L. (2008). Online-updating regularized kernel matrix factorization models for large-scale recommender systems. In *Proceedings of the ACM conference on Recommender Systems (RecSys'08)*, 251–258, ACM, New York, USA. 40

Rendle, S. & Schmidt-Thieme, L. (2010). Pairwise interaction tensor factorization for personalized tag recommendation. In *Proceedings of the 3th ACM International Conference on Web Search and Data Mining (WSDM 2010)*, 81–90, ACM, New York, USA. 86

Rendle, S., Balby Marinho, L., Nanopoulos, A. & Lars, S.T. (2009). Learning optimal ranking with tensor factorization for tag recommendation. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '09, 727–736, ACM, New York, NY, USA. 84

Rendle, S., Freudenthaler, C. & Schmidt-Thieme, L. (2010). Factorizing personalized markov chains for next-basket recommendation. In *Proceedings of the 19th International Conference on World Wide Web (WWW'10)*, 811–820, ACM, New York, USA. 84, 88

Resnick, P., Iacovou, N., Suchak, M., Bergstrom, P. & Riedl, J. (1994). Grouplens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, CSCW '94, 175–186, ACM, New York, NY, USA. 38, 47

Ricci, F., Rokach, L., Shapira, B. & Kantor, P.B., eds. (2011). *Recommender Systems Handbook*. Springer. 4

Ritter, S., Harris, T.K., Nixon, T., Dickison, D., Murray, R.C. & Towle, B. (2009). Reducing the knowledge tracing space. In *2nd International Conference On Educational Data Mining*, 151–160. 36

Rogers, J.H., Swaminathan, H. & Hambleton, R.K. (1991). *Fundamentals of Item Response Theory (Measurement Methods for the Social Science)*. Sage Publications, Inc, 1st edn. 35

Romero, C. & Ventura, S. (2006). *Data Mining in E-learning (Advances in Management Information)*. Wit Pr/Computational Mechanics. 27

Romero, C. & Ventura, S. (2010). Educational data mining: a review of the state of the art. *IEEE Transaction on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, **40**, 601–618. 27

Romero, C., Ventura, S., Espejo, P.G. & Hervs, C. (2008). Data mining algorithms to classify students. In *The 1st International Conference on Educational Data Mining (EDM 2008)*, 8–17, Montral, Canada. 27, 68

Romero, C., Ventura, S., Pechenizkiy, M. & Baker, R.S. (2010). *Handbook of Educational Data Mining*. Chapman and Hall/CRC Data Mining and Knowledge Discovery Series. 27, 36

Santos, O.C. & Boticario, J.G. (2010). Modeling recommendations for the educational domain. *Procedia Computer Science*, **1**, 2793 – 2800, proceedings of the 1st Workshop on Recommender Systems for Technology Enhanced Learning (RecSysTEL 2010). 95

Sarwar, B., Karypis, G., Konstan, J. & Reidl, J. (2001). Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web (WWW 2001)*, 285–295, ACM, New York, NY, USA. 47

Schafer, J.B., Konstan, J. & Riedi, J. (1999). Recommender systems in e-commerce. In *Proceedings of the 1st ACM conference on Electronic Commerce*, 158–166, ACM, New York, NY, USA. 4

Schein, A.I., Popescul, A., Ungar, L.H. & Pennock, D.M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th ACM International Conference on Research and Development in Information Retrieval (SIGIR 2002)*, 253–260, ACM, New York, NY, USA. 22

Shen, Y., Chen, Q., Fang, M., Yang, Q. & Wu, T. (2010). Predicting student performance : A solution for the kdd cup 2010 challenge. In *Proceedings of the KDD Cup 2010 Workshop on Improving Cognitive Models with Educational Data Mining*. 28, 68, 154

Sheng, S., Ling, C.X. & Yang, Q. (2005). Simple test strategies for cost-sensitive decision trees. In *Proceedings of the 15th European conference on Machine Learning (ECML 2005)*, 365–376. 107

Sheng, V.S. & Ling, C.X. (2006). Thresholding for making classifiers cost-sensitive. In *American Annual Conference on Artificial Intelligence (AAAI)*. 106, 117, 118

Sheng, V.S., Ling, C.X., Ni, A. & Zhang, S. (2006). Cost-sensitive test strategies. In *American Annual Conference on Artificial Intelligence (AAAI)*. 107, 117

Shute, V., Glaser, R. & Raghaven, K. (1989). Inference and discovery in an exploratory laboratory. In P. Ackerman, R. Sternberg & R. Glaser, eds., *Learning and Individual Differences*, 279–326, San Francisco: Freeman. 1

Singh, A.P. & Gordon, G.J. (2008). Relational learning via collective matrix factorization. In *Proceeding of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2008)*, KDD '08, 650–658, ACM, New York, NY, USA. 56, 58, 60

Skillicorn, D. (2007). *Understanding Complex Datasets: Data Mining with Matrix Decompositions*. Chapman and Hall/CRC. Data Mining and Knowledge Discovery Series. ISBN: 1-58488-832-6. 84

Soonthornphisaj, N., Rojsattarat, E. & Yim-ngam, S. (2006). Smart e-learning using recommender system. In *International Conference on Intelligent Computing*, 518–523. 96

Su, X. & Khoshgoftaar, T.M. (2009). A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, **2009**, 4:1–4:19. 47

Tabandeh, Y. & Sami, A. (2010). Classification of tutor system logs with high categorical features. In *Proceedings of the KDD Cup 2010 Workshop on Improving Cognitive Models with Educational Data Mining*. 28, 154

Takács, G., Pilászy, I., Németh, B. & Tikk, D. (2007). On the Gravity recommendation system. In *Proceddings of KDD Cup Workshop at SIGKDD'07, 13th*

ACM Int. Conf. on Knowledge Discovery and Data Mining, 22–30, San Jose, CA, USA. 42

Takács, G., Pilászy, I., Németh, B. & Tikk, D. (2009). Scalable collaborative filtering approaches for large recommender systems (special topic on mining and learning with graphs and relations). *Machine Learning Research*, **10**, 623–656. 41

Tan, P.N., Steinbach, M. & Kumar, V. (2005). *Introduction to Data Mining (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA. 122

Tang, T. & McCalla, G. (2004). Beyond learners interest: Personalized paper recommendation based on their pedagogical features for an e-learning system. *PRICAI 2004: Trends in Artificial Intelligence, Lecture Notes in Computer Science*, **3157**, 301–310. 95

Tang, T. & McCalla, G. (2005). Smart recommendation for an evolving e-learning system: Architecture and experiment. *International Journal on E-Learning*, **4**, 105–129. 96

Tang, Y., Zhang, Y., Chawla, N. & Krasser, S. (2009). SVMs modeling for highly imbalanced classification. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, **39**, 281–8. 107, 108

Thai-Nghe, N. (2006). *A data mining model for universities: Case studies of AIT and Cantho university*. Master's thesis, Asian Institute of Technology, Thailand. 15, 28

Thai-Nghe, N., Janecek, P. & Haddawy, P. (2007). A comparative analysis of techniques for predicting academic performance. In *Proceeding of the 37th IEEE Frontiers in Education Conference (FIE 2007)*, T2G7–T2G12, IEEE Xplore, Milwaukee, USA. 15, 28

Thai-Nghe, N., Busche, A. & Schmidt-Thieme, L. (2009). Improving academic performance prediction by dealing with class imbalance. In *Proceeding of the 9th IEEE International Conference on Intelligent Systems Design and Applications (IEEE ISDA 2009)*, 878–883, IEEE Computer Society, Pisa, Italy. 8, 17

Thai-Nghe, N., Do, T.N. & Schmidt-Thieme, L. (2010a). Learning optimal threshold for bayesian posterior probabilities to mitigate class imbalance problem. In *the 3rd International Conference on Theories and Applications of Computer Science (ICTACS 2010)*, 38 – 49. 8

Thai-Nghe, N., Do, T.N. & Schmidt-Thieme, L. (2010b). Learning optimal threshold on resampling data to deal with class imbalance. In *The IEEE International Conference on Computing and Telecommunication Technologies (IEEE RIVF 2010)*, IEEE Xplore. 8

Thai-Nghe, N., Drumond, L., Krohn-Grimberghe, A. & Schmidt-Thieme, L. (2010c). Recommender system for predicting student performance. In *Proceedings of the ACM RecSys 2010 Workshop on Recommender Systems for Technology Enhanced Learning (RecSysTEL 2010)*, vol. 1, 2811 – 2819, Elsevier's Procedia Computer Science. 7

Thai-Nghe, N., Gantner, Z. & Schmidt-Thieme, L. (2010d). Cost-sensitive learning methods for imbalanced data. In *Proceeding of the IEEE International Joint Conference on Neural Networks (IJCNN 2010). Student Travel Grant Award*, 1–8, IEEE Xplore, Barcelona, Spain. 8, 112, 117

Thai-Nghe, N., Drumond, L., Horváth, T., , Nanopoulos, A. & Schmidt-Thieme, L. (2011a). Matrix and tensor factorization for predicting student performance. In *Proceedings of the 3rd International Conference on Computer Supported Education (CSEDU 2011). Best Student Paper Award*, 69 – 78, Noordwijkerhout, the Netherlands. 7, 8

Thai-Nghe, N., Drumond, L., Horváth, T., Krohn-Grimberghe, A., Nanopoulos, A. & Schmidt-Thieme, L. (2011b). Factorization techniques for predicting student performance. In O.C. Santos & J.G. Boticario, eds., *Educational Recommender Systems and Technologies: Practices and Challenges (ERSAT 2011)*, IGI Global. 7

Thai-Nghe, N., Drumond, L., Horváth, T. & Schmidt-Thieme, L. (2011c). Multi-relational factorization models for predicting student performance. In *Proceedings of the KDD 2011 Workshop on Knowledge Discovery in Educational Data (KDDinED 2011). Held as part of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011)*, San Diego, CA, USA. 7

Thai-Nghe, N., Gantner, Z. & Schmidt-Thieme, L. (2011d). A new evaluation measure for learning from imbalanced data. In *Proceeding of the IEEE International Joint Conference on Neural Networks (IJCNN 2011). Student Travel Grant Award*, IEEE Xplore, San Jose, CA, USA. 8

Thai-Nghe, N., Horváth, T. & Schmidt-Thieme, L. (2011e). Context-aware factorization for personalized student's task recommendation. In *Proceedings of UMAP 2011 International Workshop on Personalization Approaches in Learning Environments (PALE 2011)*, CEUR-WS (ISSN 1613-0073), Girona, Spain. 8

Thai-Nghe, N., Horváth, T. & Schmidt-Thieme, L. (2011f). Factorization models for forecasting student performance. In *Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and Stamper, J. (Eds.). Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011)*, 11–20, Eindhoven, the Netherlands. 8

Thai-Nghe, N., Horváth, T. & Schmidt-Thieme, L. (2011g). Personalized forecasting student performance. In *Proceedings of the 11th IEEE International Conference on Advanced Learning Technologies (ICALT 2011)*, IEEE Computer Society, Athens, GA, USA. 7

Ting, K.M. (1998). Inducing cost-sensitive trees via instance weighting. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery*, PKDD 1998, 139–147, Springer-Verlag, London, UK. 107, 128

Ting, K.M. (2002). A study on the effect of class distribution using cost-sensitive learning. In *Discovery Science*, 98–112. 112, 115

Tomek, I. (1976). Two modifications of CNN. *IEEE Transactions on Systems, Man, and Communications SMC-6*, 769–772. 104, 108, 128, 132

Töscher, A. & Jahrer, M. (2010). Collaborative filtering applied to educational data mining. In *Proceedings of the KDD Cup 2010 Workshop on Improving Cognitive Models with Educational Data Mining*, Washington, DC, USA. 38, 66, 154

Tucker, L. (1966). Some mathematical notes on three-mode factor analysis. *Psychometrika*, 279–311. 85

Veropoulos, K., Campbell, C. & Cristianini, N. (1999). Controlling the sensitivity of support vector machines. In *Proceedings of International Joint Conference on Artificial Intelligent (IJCAI 1999)*, 55–60. 107, 117

Vozalis, E. & Margaritis, K.G. (2003). Analysis of recommender systems' algorithms. In *The 6th Hellenic European Conference on Computer Mathematics & its Applications (HERCMA), Athens, Greece*. 24, 49

Wang, B. & Japkowicz, N. (2010). Boosting support vector machines for imbalanced data sets. *Knowledge and Information Systems*, **25**, 1–20. 108, 109, 116

Wang, Q.Y., Kehrer, P., Pardos, Z. & Heffernan, N. (2011). Response tabling - a simple and practical complement to knowledge tracing. In *Proceedings of the KDD 2011 Workshop on Knowledge Discovery in Educational Data (KDDinED 2011). Held as part of the 17th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD 2011)*, San Diego, CA, USA. 36

Welch, L.R. (2003). Hidden markov models and the baum-welch algorithm. *IEEE Information Theory Society Newsletter*, **53**. 31

Wijaya, T.K. & Prasetyo, P.K. (2010). Knowledge tracing with stochastic method. In *Proceedings of the KDD Cup 2010 Workshop on Improving Cognitive Models with Educational Data Mining*, Washington, DC, USA. 36

Wilson, D.L. (1972). Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man and Cybernetics*. 104, 106

Witten, I.H. & Frank, E. (2005). *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, San Francisco, 2nd edn. 128

Woolf, B.P. (2008). *Building Intelligent Interactive Tutors, Student-Centered Strategies for Revolutionizing E-Learning*. Elsevier, Morgan Kaufmann. ix, 2, 12

Wu, J., Xiong, H., Wu, P. & Chen, J. (2007). Local decomposition for rare class analysis. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data mining*, 814–823, ACM, New York, NY, USA. 108

Xu, Y. & Mostow, J. (2011). Using logistic regression to trace multiple sub-skills in a dynamic bayes net. In *Pechenizkiy, M., Calders, T., Conati, C., Ventura, S., Romero , C., and Stamper, J. (Eds.). Proceedings of the 4th International Conference on Educational Data Mining (EDM 2011)*, 241–246. 25

Yan, R., Liu, Y., Jin, R. & Hauptmann, A. (2003). On predicting rare classes with svm ensembles in scene classification. In *The IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP 2003)*, III–21–4, IEEE Xplore. 107

Yang, P., Zhang, Z., Zhou, B.B. & Zomaya, A.Y. (2011). Sample subset optimization for classifying imbalanced biological data. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD 2011)*, 333–344, Springer. 108

Yang, Q. & Wu, X. (2006). 10 challenging problems in data mining research. *International Journal of Information Technology and Decision Making*, **5**, 597–604. 25, 102

Yu, H.F., Lo, H.Y., Hsieh, H.P., Lou, J.K., G.McKenzie, T., Chou, J.W., Chung, P.H., Ho, C.H., Chang, C.F., Wei, Y.H., Weng, J.Y., Yan, E.S., Chang, C.W., Kuo, T.T., Lo, Y.C., Chang, P.T., Po, C., Wang, C.Y., Huang, Y.H., Hung, C.W., Ruan, Y.X., Lin, Y.S., de Lin, S., Lin, H.T. & Lin, C.J. (2010). Feature engineering and classifier ensemble for kdd cup 2010. In *Proceedings of the KDD Cup 2010 Workshop on Improving Cognitive Models with Educational Data Mining*, Washington, DC, USA. 24, 28, 49, 50, 68, 78, 152, 154

Zaiane, O.R. (2002). Building a recommender agent for e-learning systems. In *Proceedings of the International Conference on Computers in Education (ICCE 2002)*, 55, IEEE Computer Society, Washington, DC, USA. 96